MaxSAT Evaluation 2021 : Solver and Benchmark Descriptions

Department of Computer Science, University of Helsinki
2021

Bacchus , F , Berg , J , Järvisalo , M & Martins , R (eds) 2021 , MaxSAT Evaluation 2021 : Solver and Benchmark Descriptions . Department of Computer Science Report Series B , vol. B-2021-2 , Department of Computer Science, University of Helsinki , Helsinki .

http://hdl.handle.net/10138/333649

unspecified
publishedVersion

MaxSAT Evaluation 2021

Solver and Benchmark Descriptions

Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins *(editors)*

University of Helsinki
Department of Computer Science
Series of Publications B
Report B-2021-2

Helsinki 2021

PREFACE

The MaxSAT Evaluations (`https://maxsat-evaluations.github.io`) are a series of events focusing on the evaluation of current state-of-the-art systems for solving optimization problems via the Boolean optimization paradigm of maximum satisfiability (MaxSAT). Organized yearly starting from 2006, the year 2021 brought on the 16th edition of the MaxSAT Evaluations, organized as a satellite event of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021). Some of the central motivations for the MaxSAT Evaluation series are to provide further incentives for further improving the empirical performance of the current state of the art in MaxSAT solving, to promote MaxSAT as a serious alternative approach to solving NP-hard optimization problems from the real world, and to provide the community at large heterogenous benchmark sets for solver development and research purposes. In the spirit of a true evaluation—rather than a competition, unlike e.g. the SAT Competition series—no winners are declared, and *no awards or medals are handed out* to overall best-performing solvers.

The 2021 evaluation consisted of a total of four tracks: two for complete solvers (one for solvers focusing on unweighted and one for solvers focusing on weighted MaxSAT instances) and two for incomplete MaxSAT solvers (using two short per-instance time limits, 60 and 300 seconds, differentiating from the per-instance time limit of 1 hour imposed in the main complete tracks). As in 2017-2020, no distinction was made between "industrial" and "crafted" benchmarks, and no track for purely randomly generated MaxSAT instances was organized.

Adhering to the new rules introduced in 2017, solvers were now required to be open-source, and the source codes of all participating solvers were made available online on the evaluation webpages after the evaluation results were presented at the SAT 2021 conference. Furthermore, a 1-2 page solver description was expected to accompany each solver submission, to provide some details on the search techniques implemented in the solvers. The solvers descriptions together with descriptions of new benchmarks for 2021 are collected together in this compilation.

Finally, we would like to thank everyone who contributed to MaxSAT Evaluation 2021 by submitting their solvers or new benchmarks. We are also grateful for the computational resources provided by the StarExec initiative which enabled running the 2021 evaluation smoothly.

Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, & Ruben Martins
MaxSAT Evaluation 2021 Organizers

Contents

Benchmarks

SOLVERS WITH MODIFICATIONS FOR 2021

CASHWMaxSAT: Solver Description

Zhendong Lei[1,2],Shaowei Cai[1,2],Dongxu Wang[3],Yongrong Peng[3],Fei Geng[3],Dongdong Wan[3],Yiping Deng[3] and Pinyan Lu[3,4]
[1]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[2]*School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China*
[3]*TCS Lab, Huawei Technologies, Shanghai, China*
[4]*Shanghai University of Finance and Economics, Shanghai, China*
{leizd, caisw}@ios.ac.cn {dongxu.wang, pengyongrong, gengfei12,wandongdong1, yiping.deng, lupinyan}@huawei.com

***Abstract*—This document describes the MaxSAT solver CASHWMaxSAT, submitted to the complete tracks(include unweighted and weighted track) of MaxSAT Evaluation 2021.**

I. INTRODUCTION

CASHWMaxSAT is a complete MaxSAT solver. It is developed from UWrMaxSat [1]. Same with UWrMaxSat, CASHWMaxSAT also applies an unsatisfiable-core-based OLL procedure [2]–[4] and uses the same constraint encoding method.

We've made four improvements based on UWrMaxSat. The first one is that we transform the maxsat problem to an Integer Programming(IP) problem and use an IP solver to solve it on both unweighted and weighted cases. The other three improvements are related to OLL procedure, and only used on unweighted cases, details will be discussed in chapter II.

CASHWMaxSAT utilizes COMiniSatPS [5] as its SAT solver and SCIP-7.0.2 [6], [7] as its IP solver.

II. DESCRIPTION

A. Utilizing an IP solver

For small-scale maxsat problem(number of variables and clauses are relatively small), we transform the maxsat problem to an IP problem, then use an IP solver to solve it in a limited time. The process detail is as following,

1) extract AtMost1 constraints from soft clauses, exactly same as UWrMaxSat
2) transform every clause(include hard and soft) to a constraint and add it to the IP solver
3) IP solver starts solving, if optimal is found, output the best solution and exit the program. Otherwise, if reaching the limited time, switch to OLL procedure

The difficulty of solving maxsat problem by OLL procedure is about increasing lower bound(LB), i.e. looking for an unsatisfiable-core by SAT solver. But IP solver could use many different algorithm(branch and bound, cutting plane, simplex, etc.) to increase LB. The idea using an IP solver is inspired by MaxHS [8].

B. MultiSolve strategy

If an unsatisfiable-core is obtained, there maybe existing more unsatisfiable-cores. As EvalMaxSAT [9] indicates, the size of the unsatisfiable-core plays an important role in the performance of OLL procedure. So we call sat solver multiple times and pick the smallest unsatisfiable-core. This strategy is inspired by EvalMaxSAT.

C. DynamicDelay strategy

In UWrMaxSat, when an unsatisfiable-core is obtained and minimized, new constraint will be encoded immediately, the variables in the unsatisfiable-core will be removed from assumptions, and the relax variable created by encoding step will be added to assumptions immediately. On the contrary, new constraint will not be encoded immediately in EvalMaxSAT.

In practice, whether delaying to encode the new constraint and add new-created relax to assumption has a big impact on OLL procedure. So we propose the DynamicDelay strategy: if the size of unsatisfiable-core is more than a predefined threshold, delaying strategy is used.

D. DelayPopOne strategy

If DynamicDelay strategy is used, when the formula becomes satisfiable, delayed relax variables will be added to assumption. We could choose adding all or only adding one. By experiments, adding one will let the problem be easy to be solved.

III. FUTURE WORK

Firstly, we could try using satlike [10], [11] to get a feasible solution and add it to IP solver's solution pool.

Secondly, we could try using sat-solver to get some unsatisfiable-cores and transform every unsatisfiable-core to a constraint, then add it to IP solver.

Finally, we could try using MultiSolve strategy, DynamicDelay strategy, DelayPopOne strategy on weighted cases.

REFERENCES

[1] M. Piotrow, "Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, (Los Alamitos, CA, USA), pp. 132–136, IEEE Computer Society, nov 2020.
[2] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub, "Unsatisfiability-based optimization in clasp," in *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.
[3] A. Ignatiev, A. Morgado, and J. Marques-Silva, "Rc2: an efficient maxsat solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 53–64, 2019.

[4] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided maxsat with soft cardinality constraints," in *International Conference on Principles and Practice of Constraint Programming*, pp. 564–573, Springer, 2014.

[5] C. Oh, *Improving SAT solvers by exploiting empirical characteristics of CDCL*. PhD thesis, New York University, 2016.

[6] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," technical report, Optimization Online, March 2020.

[7] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," ZIB-Report 20-10, Zuse Institute Berlin, March 2020.

[8] F. Bacchus, "Maxhs in the 2020 maxsat evaluation," *MaxSAT Evaluation 2020*, p. 19.

[9] F. Avellaneda, "A short description of the solver evalmaxsat," *MaxSAT Evaluation 2020*, p. 8.

[10] S. Cai and Z. Lei, "Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability," *Artif. Intell.*, vol. 287, p. 103354, 2020.

[11] Z. Lei and S. Cai, "Solving (weighted) partial maxsat by dynamic local search for sat," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 1346–1352, International Joint Conferences on Artificial Intelligence Organization, 7 2018.

A short description of the solver EvalMaxSAT

Florent Avellaneda
Computer Research Institute of Montreal
Montreal, Canada
florent.avellaneda@gmail.com

I. Introduction

EvalMaxSAT[1] is a MaxSAT solver written in modern C++ language mainly using the Standard Template Library (STL). The solver is built on top of the SAT solver Glucose [1], but any other SAT solver can easily be used instead. EvalMaxSAT is based on the OLL algorithm [2] originally implemented in the MSCG MaxSAT solver [3], [4] and then reused in the RC2 solver [5].

The OLL algorithm considers all soft variables as hard and attempts to solve the formula. If the formula has no solution, then a conjunction of soft variables that cannot be satisfied (a *core*) is extracted. Each variable constituting this core is then *relaxed* (removed from the list of soft variables or incremented if it is a cardinality) and a new cardinality is added to the list of soft variables encoding the constraint "at most one variable from the core can be false". When the formula is finally satisfied, we obtain a MaxSAT assignment.

In practice, the size of the cores plays an important role in the performance of this algorithm. Indeed, the more variables the cores contain, the more expensive the encoding of cardinalities will be. Thus, once a core is found, a core minimization phase consists of removing unnecessary variables. Although heuristics are generally used to perform this minimization, this phase remains very expensive. EvalMaxSAT performs this minimization several times by calling the solver SAT with a limited number of conflicts. In addition, the algorithm used can easily be adapted to perform the minimization in parallel with the *core* searching.

II. Description

The algorithm used is a modification of the OLL algorithm (see Algorithm 1). The main modification is that when a core is found and minimized, new variables and constraints are not added to the SAT solver immediately. All these new constraints will be added only when the formula becomes satisfiable, or when finding a new solution takes too much time. By doing that, this algorithm tries to reduce the number of implications leading from cardinality to a soft variable.

A second modification made by the EvalMaxSAT solver is in the minimize function. Indeed, this function will perform several minimizations in order to obtain small cores. A first minimization is done by making successive calls to *solver(core)* where solver calls are limited to zero conflicts. Each call to the solver attempts to remove a literal from the core (a literal can be removed if the formulation remains unsatisfiable). After that, we apply the same algorithm with 1000 limited conflicts by considering the variables in differing orders.

[1]See https://github.com/FlorentAvellaneda/EvalMaxSAT

Algorithm 1 (Pseudo-code of the sequential algorithm)

Input: A formula φ

1: $cost \leftarrow extractAM1(\varphi)$
2: **while** $true$ **do**
3: $(st, \varphi_c) \leftarrow SATSolver(\varphi)$
4: **if** $st = true$ **then**
5: $\varphi \leftarrow \varphi \cup \varphi_{tmp}$
6: $(st, \varphi_c) \leftarrow SATSolver(\varphi)$
7: **if** $st = true$ **then**
8: **return** $cost$
9: **end if**
10: **end if**
11: $\varphi_c \leftarrow minimize(\varphi, \varphi_c)$
12: $k \leftarrow exhaust(\varphi, \varphi_c)$
13: $cost \leftarrow cost + k$
14: $\varphi \leftarrow relax(\varphi, \varphi_c)$
15: $\varphi_{tmp} \leftarrow \varphi_{tmp} \cup createSum(\varphi_c, k)$
16: **end while**

III. Implementation Details

Extract AM1 Two algorithms are used to extract AtMost1 constraints from soft variables. The first one uses the mcqd library [6] to find the maximum clique in the incompatibility graph of the soft variables; the second one uses a heuristic.

Cardinality The Totalizer Encoding [7] is used to represent cardinalities. The implementation reuses the code from the PySAT's ITotalizer [8].

Exhaust After the minimization is performed, a *core exhaustion* [5] or *cover optimization* [9] is done.

Timeout Many timeouts are used to stop minimization when they take too much time.

IV. Multicore version

An interesting feature of the algorithm used is that it is very simple to parallelize it. Although the competition does not allow the multi-threaded calculation, this feature has been implemented in the solver **but disabled for the competition**. The architecture of the parallelized algorithm is depicted in Figure 1. The main thread looks for new cores to minimize and when it finds one, it removes all variables

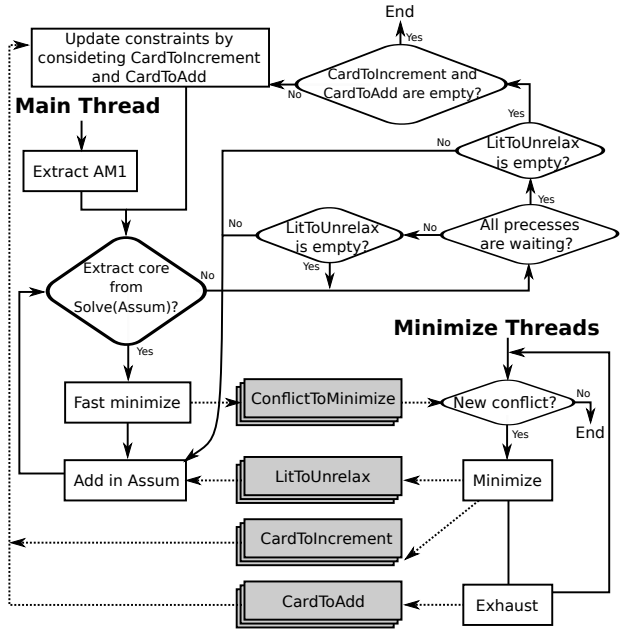

Fig. 1. Algorithm architecture

present in the core from the list of soft variables (Assum). Paralleling this, threads access the cores found by the main thread (ConflictToMinimize), minimize them and share new cardinalities (CardToAdd), unused variables (LitToUnrelax) and cardinalities to be incremented (CardToIncrement). Before searching for new cores, the main thread collects the variables that had previously been removed but were not used in any previous thread.

When the main thread no longer finds a core, all minimization threads have been completed and no variables are to be reconsidered as soft, then the main thread considers the new cardinalities to be added and incremented before restarting the *core* search. If there are no cardinalities to add or increment, then the search is complete and we get a MaxSAT assignment.

REFERENCES

[1] G. Audemard, J. Lagniez, and L. Simon, "Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, ser. Lecture Notes in Computer Science, M. Järvisalo and A. V. Gelder, Eds., vol. 7962. Springer, 2013, pp. 309–317. [Online]. Available: https://doi.org/10.1007/978-3-642-39071-5_23

[2] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, ser. Lecture Notes in Computer Science, B. O'Sullivan, Ed., vol. 8656. Springer, 2014, pp. 564–573. [Online]. Available: https://doi.org/10.1007/978-3-319-10428-7_41

[3] A. Morgado, A. Ignatiev, and J. Marques-Silva, "MSCG: robust core-guided maxsat solving," *J. Satisf. Boolean Model. Comput.*, vol. 9, no. 1, pp. 129–134, 2014. [Online]. Available: https://satassociation.org/jsat/index.php/jsat/article/view/127

[4] A. Ignatiev, A. Morgado, V. M. Manquinho, I. Lynce, and J. Marques-Silva, "Progression in maximum satisfiability," in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, ser. Frontiers in Artificial Intelligence and Applications, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds., vol. 263. IOS Press, 2014, pp. 453–458. [Online]. Available: https://doi.org/10.3233/978-1-61499-419-0-453

[5] A. Ignatiev, A. Morgado, and J. Marques-Silva, "RC2: an efficient maxsat solver," *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, 2019. [Online]. Available: https://doi.org/10.3233/SAT190116

[6] J. Konc and D. Janezic, "An improved branch and bound algorithm for the maximum clique problem," *proteins*, vol. 4, no. 5, 2007.

[7] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce, "Reflections on "incremental cardinality constraints for maxsat"," *CoRR*, vol. abs/1910.04643, 2019. [Online]. Available: http://arxiv.org/abs/1910.04643

[8] A. Ignatiev, A. Morgado, and J. Marques-Silva, "Pysat: A python toolkit for prototyping with SAT oracles," in *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science, O. Beyersdorff and C. M. Wintersteiger, Eds., vol. 10929. Springer, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26

[9] C. Ansótegui, M. L. Bonet, J. Gabas, and J. Levy, "Improving wpm2 for (weighted) partial maxsat," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2013, pp. 117–132.

MaxSAT Evaluation 2021: Solver and Benchmark Descriptions, volume B-2021-2 of Department of Computer Science Series of Publications B, University of Helsinki 2021.

Exact: evaluating a pseudo-Boolean solver on MaxSAT problems

Jo Devriendt
KU Leuven
Leuven, Belgium
jo.devriendt@kuleuven.be

***Abstract*—Weighted MaxSAT solving is a special case of pseudo-Boolean optimization, also known as binary linear programming. This submission aims to investigate whether Exact, a conflict-driven cutting planes learning pseudo-Boolean solver, is competitive on MaxSAT problems.**

***Index Terms*—binary linear programming, pseudo-Boolean solving, cutting planes, core-guided optimization**

I. Introduction

It is well-known[1] that a weighted MaxSAT formula can be written as a binary linear program (BLP):

$$\begin{aligned} \text{Minimize} \quad & \sum_{c\in C} w_c z_c \\ \text{s.t.} \quad & z_c + \sum_{x\in c^+} x + \sum_{y\in c^-} (1-y) \geqslant 1 \qquad \forall c \in C \end{aligned}$$

where C is a set of clauses, c^+ and c^- are the set of positive and negative literals in a clause $c \in C$ respectively, w_c is the cost of not satisfying c, and all variables x, y and z are binary.

Even though this BLP formulation is natural, the state-of-the-art in previous MaxSAT evaluations employs repeated calls to Boolean satisfiability (SAT) solvers instead of one straightforward call to an integer linear programming (ILP) solver. Most likely, the reason for this is that ILP solvers rely heavily on exploiting the linear relaxation of a BLP, while all constraints in the above BLP are clauses, which have a particularly weak linear relaxation.

A third technology that could natively handle the above BLP however is pseudo-Boolean (PB) solving. Similar to ILP technology, PB technology natively takes linear constraints over binary variables as input. However, in contrast to ILP solvers, a PB solver does not chiefly depend on reasoning on the linear relaxation of a BLP. Instead, so-called *conflict-driven cutting-planes learning* (CDCPL) PB solvers derive (*learn*) from each conflict in the search tree an implied linear constraint that, if it had been derived previously, would have prevented the conflict through unit propagation. In this way, CDCPL PB solvers are a generalization of *conflict-driven clause learning* (CDCL) SAT solvers, where a CDCPL solver can learn stronger constraints than clauses.

[1]See, e.g., https://en.wikipedia.org/wiki/Maximum_satisfiability_problem#(1-1/e)-approximation

II. Submission

We submit the CDCPL solver Exact[2] to the MaxSAT evaluation. Exact is a fork of the CDCPL solver RoundingSat[3] [1]. For this submission, we do not employ RoundingSat's linear programming integration [2], as we expect the linear relaxations of the instances to be too weak. We do make use of its optimized propagation routines [3] and its hybrid core-guided optimization technique [4].

Exact improves upon its predecessor through a myriad of refactorings, extensions and improvements. We highlight three important ones for this MaxSAT evaluation submission.

A first one is the *stratification* routine of Exact's core-guided optimization. Instead of core-guided stratification based on [5], Exact uses a simple routine that ignores all soft clauses with a cost lower than some τ, which initially is set to the highest clause cost (the highest weight in the objective of the BLP representation). If Exact does not find a core with this τ (i.e., it finds a solution where all hard and non-ignored soft clauses are satisfied, or timeouts in the core-guided search) τ is halved, to consider more soft clauses. This process is repeated until the maximum cost is halved to 1, at which point all soft clauses are taken into account.

A second improvement is the exploitation of the observation that a single *PB core* may yield multiple *cardinality cores*, which can be used during the core-guided lower bound derivation and objective reformulation process [4]. For instance, given an objective function $4x+3y+2z+w$ to be minimized, and a PB core $2x + 2y + z + w \geqslant 4$, Exact constructs an initial implied cardinality core $x + y + z \geqslant 2$, reformulating the objective to $2x + y + w + 2a + 4$ through the *extension constraint* $x + y + z = 2 + a$. But as $2x + 2y + z + w \geqslant 4$ also implies $x + y + w \geqslant 2$, Exact can further reformulate the objective to $x + 2a + b + 6$ with the extension constraint $x + y + w = 2 + b$, increasing the objective lower bound from 4 to 6 without any new core-guided solver call.

A third improvement is meant to address the fact that, given a search conflict implied by only clausal constraints, CDCPL solvers can only learn a clause, which is identical to regular CDCL SAT solving (which has a more efficient implementation). For CDCPL to work well, non-clausal constraints need to appear in the conflict implication graph, so that

[2]https://gitlab.com/JoD/exact
[3]https://gitlab.com/miao_research/roundingsat

strong non-clausal constraints can be learned [6]. On MaxSAT instances, Exact introduces non-clausal constraints in three ways. Firstly, a derived upper or lower bound on the objective function is typically a non-clausal constraint. Secondly, a core-guided extension constraint also typically is equivalent to a conjunction of non-clausal constraints. Thirdly, implied cardinality constraints can be detected from a conjunction of clauses. Work on cardinality detection in RoundingSat exists [7], where an investigation of the implication graph during conflict analysis yields the right information to construct cardinality constraints. Exact uses a different approach, where repeated *probing* (deciding a single variable and running unit propagation) yields the necessary edges in the implication graph to derive *at-most-one* cardinality constraints.

III. Conclusion

By combining the effectiveness of CDCLP and core-guided optimization, PB solving technology may have become competitive to SAT-based approaches on MaxSAT problems. Exact's submission to 2021's MaxSAT evaluation will provide experimental data to support or reject this hypothesis.

IV. Acknowledgment

Exact is a fork of RoundingSat, which in turn uses code from MiniSat [8]. The author of Exact is Jo Devriendt (KU Leuven). The authors of RoundingSat are Jan Elffers (formerly Lund University), Jo Devriendt (KU Leuven), Stephan Gocht (Lund University) and Jakob Nordström (University of Copenhagen, Lund University). The authors of MiniSat are Niklas Eén (formerly University of California) and Niklas Sörensson (formerly Chalmers University of Technology).

References

[1] J. Elffers and J. Nordström, "Divide and conquer: Towards faster pseudo-Boolean solving," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, Jul. 2018, pp. 1291–1299.

[2] J. Devriendt, A. Gleixner, and J. Nordström, "Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search," in *Proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '20)*, ser. Lecture Notes in Computer Science, vol. 12296. Springer, Sep. 2020, pp. xxiv–xxv.

[3] J. Devriendt, "Watched propagation of 0-1 integer linear constraints," in *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, ser. Lecture Notes in Computer Science, vol. 12333. Springer, Sep. 2020, pp. 160–176.

[4] J. Devriendt, S. Gocht, E. Demirović, J. Nordström, and P. Stuckey, "Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning," vol. 35, no. 5, May 2021, pp. 3750–3758. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16492

[5] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving SAT-based weighted MaxSAT solvers," in *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, ser. Lecture Notes in Computer Science, vol. 7514. Springer, Oct. 2012, pp. 86–101.

[6] S. R. Buss and J. Nordström, "Proof complexity and SAT solving," in *Handbook of Satisfiability*, 2nd ed., ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, Feb. 2021, vol. 336, ch. 7, pp. 233–350.

[7] J. Elffers and J. Nordström, "A cardinal improvement to pseudo-Boolean solving," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, Feb. 2020, pp. 1495–1503.

[8] N. Eén and N. Sörensson, "An extensible SAT-solver," in *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03), Selected Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2004, pp. 502–518.

MaxHS in the 2021 MaxSat Evaluation

Fahiem Bacchus
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
Email: fbacchus@cs.toronto.edu

I. MaxHS

MaxHS originated in the work of Davies [1] who developed the first MaxSat solver based on the Implicit Hitting Set approach (IHS). The core components of MaxHS are described in [1]–[4]. The PhD thesis of Saikko [5] also provides an excellent overview of the IHS approach along with a number of additional insights. In addition to various algorithmic and code improvements over the years, MaxHS also employs the techniques of reduced cost fixing [6] and abstract cores [7]. Both of these techniques go beyond the basic IHS approach.

II. 2021

As with the 2020 MaxHS entry to the 2021 entry detects and utilizes abstract cores when these are useful. In 2021 some more tuning of when to trigger the construction of abstract cores was done. These changes added to the solver's robustness but did not do much to enhance its performance. It was noticed however, that extracting abstract cores is more difficult for the SAT solver than extracting ordinary cores. It was also noted that the size of the new MaxSat instances submitted to the competition was growing, again increasing the burden on the SAT solver. In the 2020 version of MaxHS utilized MiniSat v2.2 as its SAT solver. This version had been modified in minor ways to improve its effectiveness on the problems MaxHS required it to solve. But it was clear that it was time to move on to a more effective SAT solver.

Interesting, in previous years we had tested replacing MiniSat with Glucose in MaxHS, but MaxHS with Glucose had always performed slightly worse. So a decision was made to move to the non-MiniSat based solver Cadical [8]. This change yielded better performance for MaxHS (albeit on somewhat limited testing).

References

[1] J. Davies, "Solving maxsat by decoupling optimization and satisfaction," Ph.D. dissertation, University of Toronto, 2013.

[2] J. Davies and F. Bacchus, "Solving MAXSAT by Solving a Sequence of Simpler SAT Instances," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 6876. Springer, 2011, pp. 225–239.

[3] ——, "Exploiting the power of mip solvers in maxsat," in *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, ser. Lecture Notes in Computer Science, M. Järvisalo and A. V. Gelder, Eds., vol. 7962. Springer, 2013, pp. 166–181. [Online]. Available: https://doi.org/10.1007/978-3-642-39071-5_13

[4] ——, "Postponing optimization to speed up MAXSAT solving," in *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, ser. Lecture Notes in Computer Science, C. Schulte, Ed., vol. 8124. Springer, 2013, pp. 247–262.

[5] P. Saikko, "Implicit hitting set algorithms for constraint optimization," Ph.D. dissertation, University of Helsinki, 2019, https://helda.helsinki.fi/bitstream/handle/10138/306951/implicit.pdf?sequence=1&isAllowed=y.

[6] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko, "Reduced cost fixing in maxsat," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 10416. Springer, 2017, pp. 641–651. [Online]. Available: https://doi.org/10.1007/978-3-319-66158-2_41

[7] J. Berg, F. Bacchus, and A. Poole, "Abstract cores in implicit hitting set maxsat solving," in *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, ser. Lecture Notes in Computer Science, L. Pulina and M. Seidl, Eds., vol. 12178. Springer, 2020, pp. 277–294. [Online]. Available: https://doi.org/10.1007/978-3-030-51825-7_20

[8] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

Open-WBO @ MaxSAT Evaluation 2021

Ruben Martins
rubenm@cs.cmu.edu
CMU, USA

Norbert Manthey
nmanthey@conp-solutions.com
Dresden, Germany

Miguel Terra-Neves, Vasco Manquinho, Inês Lynce
{neves,vmm,ines}@inesc-id.pt
INESC-ID/IST, Portugal

I. Introduction

OPEN-WBO [1] is an open source MaxSAT solver that supports several MaxSAT algorithms [2], [3], [4], [5], [6], [7], [8] and SAT solvers [9], [10], [11]. OPEN-WBO is particularly efficient for unweighted MaxSAT and has been one of the best solvers in the MaxSAT Evaluations from 2014 to 2017. Two versions of OPEN-WBO were submitted to the unweighted track at MaxSAT Evaluation 2021: open-wbo-res-mergesat and open-wbo-res-glucose. The only difference between Open-WBO 2020 and 2021 versions is a newer version of the mergesat SAT solver [11]. The remainder of this document describes the differences between these versions.

II. SAT solvers

OPEN-WBO is based on the data structures of MINISAT 2.2 [9], [12]. Therefore, solvers based on MINISAT 2.2 can be used as a potential back-end solver. For the MaxSAT Evaluation 2021, we use GLUCOSE 4.1 [10], [13], [14] as the back-end SAT solver of the version that ends in glucose and MERGESAT [11] as the back-end SAT solver of the version that ends in mergesat.

MERGESAT [11] is a new CDCL solver developed by Norbert Manthey and it is based on the SAT competition winner of 2018, MAPLELCMDISTCHRONOBT [15], and adds several known techniques. For restarts, only partial backtracking is used, learned clause minimization is implemented more efficiently, and also applies simplification again in case the first swipe resulted in a simplification. The time-based decision heuristic switch is made deterministic by using solving steps. Assumption literals are set before search, and the CCNR SLS engine, as well as polarity selection during decision with rephasing is used. To support being used inside MaxSAT solvers, the incremental search feature had to be enabled again.

III. MaxSAT Algorithms

In this section, we briefly describe the algorithms used for the complete track at the MSE2021.

A. Complete Unweighted Track

Two versions were submitted to the complete unweighted track: open-wbo-res-mergesat and open-wbo-res-glucose.

Both versions use a variant of the unsatisfiability-based algorithm MSU3 [3] and the OLL algorithm [7]. This algorithm works by iteratively refining a lower bound λ on the number of unsatisfied soft clauses until an optimum solution is found. We use an incremental version of this algorithm by taking advantage of the incremental version of the Totalizer encoding [4]. We also extended the incremental MSU3 algorithm [4] with resolution-based partitioning techniques [8]. We represent a MaxSAT formula using a resolution-based graph representation and iteratively join partitions by using a proximity measure extracted from the graph representation of the formula. The algorithm ends when only one partition remains and the optimal solution is found. Since the partitioning of some MaxSAT formulas may be unfeasible or not significant, we heuristically choose to run either MSU3 with partitions or without partitions. In particular, we do not use partition-based techniques when one of the following criteria is met: (i) the formula is too large ($>$ 1,000,000 clauses), (ii) the ratio between the number of partitions and soft clauses is too high ($>$ 0.8), (iii) the sparsity of the graph is too small ($<$ 0.04), or (iv) there exist some at-most-one relations between soft clauses ($>$ 10), i.e. if one soft clause is satisfied it implies that some other soft clauses will be unsatisfied.

B. Preprocessing

We perform identification of unit cores and at-most-one relations between soft clauses by using unit propagation. A similar technique is done in RC2 [16], the winner of the MaxSAT Evaluation 2018.

C. Other

OPEN-WBO now supports printing the certificate in a compact mode using 0's and 1's.

IV. Availability

The latest release of OPEN-WBO is available under a MIT license in GitHub at *https://github.com/sat-group/open-wbo*.

Acknowledgments

We would like to thank Laurent Simon and Gilles Audemard for allowing us to use GLUCOSE 4.1 in the MaxSAT Evaluation. We would also like to thank Niklas Eén and Niklas Sörensson for the development of MINISAT 2.2. Additionally, we would like to thank all the collaborators on previous versions of OPEN-WBO, namely Saurabh Joshi and Mikoláš Janota. Finally, we would like to thank David Chen for his study on the impact of disjoint cores, unit cores, and at-most-one relations between soft clauses that were done in the scope of Independent Studies at CMU.

REFERENCES

[1] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.

[2] V. Manquinho, J. Marques-Silva, and J. Planes, "Algorithms for Weighted Boolean Optimization," in *SAT*. Springer, 2009, pp. 495–508.

[3] J. Marques-Silva and J. Planes, "On Using Unsatisfiability for Solving Maximum Satisfiability," *CoRR*, 2007.

[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.

[5] R. Martins, V. Manquinho, and I. Lynce, "On Partitioning for Maximum Satisfiability," in *ECAI*. IOS Press, 2012, pp. 913–914.

[6] R. Martins, V. M. Manquinho, and I. Lynce, "Community-based partitioning for maxsat solving," in *SAT*. Springer, 2013, pp. 182–191.

[7] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-Guided MaxSAT with Soft Cardinality Constraints," in *CP*. Springer, 2014, pp. 564–573.

[8] M. Neves, R. Martins, M. Janota, I. Lynce, and V. M. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.

[9] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.

[10] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.

[11] N. Manthey, "The SAT solver MergeSat," in *SAT*. Springer, 2021.

[12] N. Sörensson, N. Een, and N. Manthey. (2018, May) GitHub repository for MiniSat. https://github.com/conp-solutions/minisat.

[13] G. Audemard, J.-M. Lagniez, and L. Simon, "Improving glucose for incremental sat solving with assumptions: Application to mus extraction," in *SAT*. Springer, 2013.

[14] G. Audemard and L. Simon. (2018, May) Glucose's home page. http://www.labri.fr/perso/lsimon/glucose.

[15] A. Nadel and V. Ryvchin, "Chronological backtracking," in *SAT*. Springer, 2018, pp. 111–121.

[16] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python Toolkit for Prototyping with SAT Oracles," in *Proc. SAT*, ser. Lecture Notes in Computer Science, O. Beyersdorff and C. M. Wintersteiger, Eds., vol. 10929. Springer, 2018, pp. 428–437.

UWrMaxSat in MaxSAT Evaluation 2021

Marek Piotrów
Institute of Computer Science, University of Wrocław
Wrocław, Poland
marek.piotrow@uwr.edu.pl

***Abstract*—UWrMaxSat is a complete solver that can be used to solve not only partial weighted MaxSAT instances but also pseudo-Boolean ones. It can be also characterized as anytime solver, because it outputs the best known solution, when its run is interrupted. It incrementally uses COMiniSatPS by Chanseok Oh (2016) as an underlying SAT solver, but may be compiled with a few other solvers as well. In its main configuration, UWrMaxSat applies a core-guided OLL procedure, where it uses own sorter-based pseudo-Boolean constraint encoding to translate cardinality constraints into CNF. It can switch to a binary search strategy after a fixed number of conflicts and then it uses an improved encoding of a pseudo-Boolean goal function, where different bounds on its value are set only by assumptions.**

***Index Terms*—MaxSAT-solver, UWrMaxSat, COMiniSatPS, sorter-based encoding, core-guided, complete solver**

I. Introduction

A short history of the solver is as follows: In 2018 Michał Karpiński and Marek Piotrów created a new pseudo-Boolean (PB) constraint solver called KP-MiniSat+ [7] as an extension of MiniSat+ 1.1 solver by Eén and Sörensson (2012) [5]. In the solver we replaced the encoding based on odd-even sorting networks by a new one using our construction of selection networks called 4-Way Merge Selection Networks [8]. We also optimized mixed-radix base searching procedure and added a few other optimizations based on literature. In 2020 the encoding was extended in such a way that a goal function is encoded only once and then SAT-solver assumptions are used to set different bounds on its value. Our experiments showed that the solver is competitive to other state-of-art PB solvers.

At the end of 2018, KP-MiniSat+ was extended to deal with MaxSAT instances and renamed to UWrMaxSat. Three different solving techniques for MaxSAT problems were added to UWrMaxSat together with a translation of PB instances to MaxSAT ones, and vice-versa. In 2019 and 2020 the solver was submitted to MaxSAT Evaluations, where it won the complete-weighted track in 2020 and was ranked second places in both complete-weighted and complete-unweighted tracks in 2019.

This year a new version is submitted to the evaluation with two main additional features: (1) a greedy algorithm is added to the encoder module to check if some of previously encoded sorting networks can be reused in a new encoding, and (2) an algorithm is implemented in the solver module to detect, so called, generalized Boolean multilevel optimization (GBMO) splitting points, defined in [12].

II. Description

This year version of UWrMaxSat is denoted as 1.2. For the main features of versions 1.0 and 1.1 see [13], [14]. A more detailed description of UWrMaxSat ver. 1.1 can be found in [15]. In the current version, we continue to use incrementally COMiniSatPS by Chanseok Oh (2016) [11] as an underlying SAT solver. The default search strategy for the optimal solution is also the same as in previous years, that is, a core-guided linear unsat-sat one, where unsatisfiability cores are processed by the OLL procedure [1], [6], [9] and cardinality constraints generated by it are encoded with the help of 4-Way Merge Selection Networks [8] and Direct Networks [4]. If the linear unsat-sat searching is unsuccessful after a predefined number of conflicts, it can be switched to a binary search technique similar to that of the original MiniSat+ [5], without restarting the SAT solver. Note that previously it was done after a predefined number of seconds. We change this to make the solver less hardware dependent. The general description of search strategies used by MaxSAT solvers can be found, for example, in [10].

It is well known, that a sorter-based encoder gets an input sequence of literals $(x_1, \ldots, x_n)$ and its task is to produce a new sequence of literals $(y_1, \ldots, y_n)$ and a set of clauses such that, for any $k \in \{1, \ldots, n\}$, whenever any subset of inputs of size k is set to 1 by the SAT solver, its unit propagation procedure forces all $y_1, \ldots, y_k$ to be also 1. Note that the order of literals in $(x_1, \ldots, x_n)$ is not important in this task, so the encoder can change it. In the process of solving, the encoder is called many times with different input sequences and it can happen that subsets of the current input have already been encoded. Such a situation is detected in our improved encoder and the corresponding clauses are reused. A greedy strategy with respect to the size of a subsets is implemented to reduce the complexity of the algorithm. The impact of this improvement was mainly observed in solving PB instances.

GBMO was first defined and explored by Paxian, Raiola and Becker in [12], as a preprocessing technique for weighted MaxSAT instances. We have implemented a detection procedure of GBMO splitting points in the MaxSAT solving module, but the detected ones are not fully explored yet in our solving algorithm (due to time limitations). We just used them in the stratification heuristic [2], [3] to harden a corresponding soft constraint. We are planning to use them in all solving strategies, in particular, in the binary-search one.

Finally, the parser of UWrMaxSat was modified to accept

the new proposed input format of partial weighted MaxSAT instances. Recall that there is no p-line in it and hard clauses are preceded by the letter 'h' instead of the "very big" weight given in a p-line.

REFERENCES

[1] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)*, volume 17, pages 212–221, 2012.

[2] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In Michela Milano, editor, *Proc. CP*, pages 86–101. Springer, 2012.

[3] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77 – 105, 2013.

[4] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[5] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[6] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Rc2: an efficient maxsat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 53–64, 2019.

[7] Michał Karpiński and Marek Piotrów. Competitive sorter-based encoding of pb-constraints into sat. In Daniel Le Berre and Matti Järvisalo, editors, *Proc. Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 65–78. EasyChair, 2019.

[8] Michał Karpiński and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints*, Apr 2019.

[9] Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Proc. CP*, LNCS, pages 564–573. Springer, 2014.

[10] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[11] Chanseok Oh. *Improving SAT Solvers by Exploiting Empirical Characteristics of CDCL*. PhD thesis, New York University, 2016.

[12] Tobias Paxian, Pascal Raiola, and Bernd Becker. On preprocessing for weighted maxsat. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 556–577, Cham, 2021. Springer International Publishing.

[13] Marek Piotrów. Uwrmaxsat - a new minisat+-based solver in maxsat evaluation 2019. In Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, editors, *MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions*, Department of Computer Science Report Series B-2019-2, pages 11–12, 2019.

[14] Marek Piotrów. Uwrmaxsat - an efficient solver in maxsat evaluation 2020. In Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins, editors, *MaxSAT Evaluation 2020 : Solver and Benchmark Descriptions*, Department of Computer Science Report Series B-2020-2, pages 34–35, 2020.

[15] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136, 2020.

SATLike-c: Solver Description

Zhendong Lei[1,2],Shaowei Cai[1,2],Fei Geng[3],Dongxu Wang[3],Yongrong Peng[3],Dongdong Wan[3],Yiping Deng[3] and Pinyan Lu[3,4]
[1]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[2]*School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China*
[3]*TCS Lab, Huawei Technologies, Shanghai, China*
[4]*Shanghai University of Finance and Economics, Shanghai, China*
{leizd, caisw}@ios.ac.cn {gengfei12, dongxu.wang, pengyongrong, wandongdong1, yiping.deng, lupinyan}@huawei.com

***Abstract*—This document describes the solver SATLike-c, submitted to the four incomplete tracks of MaxSAT Evaluation 2021.**

I. Introduction

SATLike-c participates in incomplete track. SATLike-c has two main engines, one is local search solver SATLike [1] and the other is SAT-based solver TT-Open-WBO-inc [2].

A. Local Search Algorithm: SATLike

SATLike adopts a dynamic local search framework for SAT and exploits the distinction of hard and soft clauses by a carefully designed clauses weighting scheme. The clauses weighting scheme works on both hard and soft clauses while it puts more increments to hard clauses each time and also sets a limit on the maximum weight that each soft clause can get. As for the variable selection heuristic, it works like a normal local search for SAT which pick a variables with the highest score in each step. The algorithm is thus called SATLike.

The weighting scheme used in SATLike is named **Weighting-PMS**, and works as follows. For each hard clause, we associate an integer number as its weight which is initialized to 1; for each soft clause, we use the original weight (which is 1 for PMS, and is the original weight from the input file for WPMS) as its initial weight. Whenever a "stuck" situation is observed, that is, we cannot decrease the cost by flipping any variable, then clause weights are updated as follows.

- with probability $1 - sp$: for each falsified hard clause c, $w(c) := w(c) + h\_inc$; for each falsified soft clause c, $w(c) := w(c) + 1$ if $w(c) < \zeta$, where ζ limits the maximum value that a soft clause weight can get.
- with probability sp (smoothing probability): for each satisfied hard clause c s.t. $w(c) > 1$, $w(c) := w(c) - h\_inc$; for each satisfied soft clause c s.t. $w(c) > 1$, $w(c) := w(c) - 1$.

SATLike uses scoring function (the score of variables) to guide the search. In SATLike, the score of variable x, denoted by $score(x)$, is the increase of total weight of satisfied clauses (either hard clauses or soft clauses) caused by flipping x.

The main component of SATLike is a loop (lines 3-15), which is executed to iteratively modify the current solution α until a given time limit is reached. During the search, whenever a better feasible solution is found, the best feasible solution is updated accordingly (line 4).

Algorithm 1: SATLike

Input: PMS instance F, *cutoff*
Output: A feasible assignment α of F and its cost, or "no feasible assignment found"

```
1  begin
2      α := an initial complete assignment; α* := ∅;
3      while elapsed time < cutoff do
4          if ∄ falsified hard clauses & cost(α) < cost* then
               α* := α; cost* := cost(α) ;
5          if D := {x|score(x) > 0} ≠ ∅ then
6              v := a variable in D picked by BMS strategy;
7          else
8              update weights of clauses by Weighting-PMS;
9              if ∃ falsified hard cluases then
10                 c := a random falsified hard clause
11             else c := a random falsified soft clause;
12             v :=the variable with highest score in c;
13         α := α with v flipped;
14     if α* is feasible then return (cost*, α*);
15     else return "no feasible assignment found";
```

In each step, if there exits variables with score bigger than 0, SATLike picks a variable with the greatest score and flips it. If there is no such variable, then *SATLike* updates clause weights according to the Weighting-PMS, and picks a variable from a falsified clause.

B. Hybrid Solver: SATLike-c

We combine SATLike with the state of the art SAT-based solvers TT-Open-WBO-inc [2], which leading to the hybrid solver SATLike-c.

The structure of SATLike-c is shown as algorithm 2. First, a SAT solver is executed to find a feasible solution (only works on hard clauses). Then SATLike is executed with this feasible solution as its initial solution. SATLike is executed until there is no improvement over k steps (k is set to 10^7 in our experiment). In most cases of our solver, SATLike can return a high-quality solution in this period, which is even close to the optimal one. But as the execution time increases, it is difficult for SATLike to get a further improved solution. So, the obtained high-quality solution is passed to the SAT-based

Algorithm 2: SATLike-c

Input: PMS instance F, *cutoff*
Output: A feasible assignment α of F and its cost, or "no feasible assignment found"

```
begin
    F'= Hard(F);
    α := SATSOLVER(F');
    α := SATLIKE(α, F);
    α := TTOPENWBOINC(α, F);
    if α* is feasible then return (cost*, α*);
    else return "no feasible assignment found";
```

solver as the initial model, and thus an initial upper bound is also provided. After that, TT-Open-WBO-inc is executed in the rest time.

REFERENCES

[1] Shaowei Cai, Zhendong Lei, "Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability". Artif. Intell. 287: 103354 (2020)

[2] Alexander Nadel."Anytime weighted maxsat withimproved polarity selection and bit-vector optimization" InClark W. Barrett and Jin Yang, editors,2019 Formal Methodsin Computer Aided Design, FMCAD 2019, San Jose, CA, USA,October 22-25, 2019, pages 193–202. IEEE, 2019

TT-Open-WBO-Inc-21: an Anytime MaxSAT Solver Entering MSE'21

Alexander Nadel
Email: alexander.nadel@cs.tau.ac.il

***Abstract*—This document describes the solver TT-Open-WBO-Inc-21, submitted to the four incomplete tracks of MaxSAT Evaluation 2021. TT-Open-WBO-Inc-21 is the 2021 version of our solver TT-Open-WBO-Inc [9], [10] (itself based on Open-WBO-Inc [3]), which came in first in both the incomplete weighted tracks and second in both the incomplete unweighted tracks at MaxSAT Evaluation 2020 (MSE20). TT-Open-WBO-Inc-21 includes the following two major new features as compared to the previous version TT-Open-WBO-Inc-20: 1) integration of SATLike [2] for inprocessing, and 2) further modifications (as compared to TT-Open-WBO-Inc-20 [10]) to the Polosat algorithm [8].**

I. Introduction

Applying the SATLike local search algorithm [2] as a preprocessor, followed by invoking a SAT-based anytime MaxSAT algorithm proved to be a successful strategy for anytime MaxSAT solving, used by both the winner of MSE20 in the two incomplete unweighted tracks SATLike-c-20 [4] and the runner-up in the two incomplete weighted tracks SATLike-cw-20 [5].

Following these results, we have integrated SATLike into TT-Open-WBO-Inc. Moreover, while the weighted component of our solver uses SATLike as a preprocessor, the unweighted component uses it as an *inprocessor*, that is, SATLike is invoked more than once during the solver's lifetime. See Sect. II below for more details.

Polosat algorithm [8] can be understood as a SAT-based local search. Using it *instead* of plain SAT solving significantly improves the performance of anytime MaxSAT solving [8]. TT-Open-WBO-Inc-20 had used Polosat or, more precisely, a modified version of the algorithm, for both weighted and unweighted solving. In our new version TT-Open-WBO-Inc-21, we changed Polosat further, where the modifications differ between the weighted and the unweighted components. Sect. III contains further details.

II. Integrating SATLike

We found in preliminary experiments that it pays off to integrate SATLike in a different manner into the weighted and unweighted components of the solver, respectively.

A. SATLike *in the Weighted Component*

The weighted component of the solver uses SATLike as part of its flow as follows (similarly to the way it is used by SATLike-c-20 and SATLike-cw-20):

1) Run a SAT solver to find an initial model μ.
2) Invoke SATLike for 15 seconds to improve μ, if possible.
3) Switch to an anytime SAT-based algorithm, where μ is used as the initial model. For the anytime SAT-based algorithm, we apply the BMO-based clustering [3] with TORC polarity selection [6], in which SAT invocations are replaced by invocations of our enhanced version of Polosat, discussed in Sect. III-A.

B. SATLike *in the Unweighted Component*

The unweighted component applies SATLike for inprocessing as part of the following flow:

1) Run a SAT solver to find an initial model μ.
2) Invoke SATLike for 15 seconds to improve μ, if possible.
3) Switch to an anytime SAT-based algorithm, where μ is used as the initial model. We apply the Mrs. Beaver algorithm [7], enhanced by TORC polarity selection [6] and two further heuristics from [6]: global stopping condition for OBV-BS and size-based switching to complete part. The SAT invocations are replaced by invocations of our enhanced version of Polosat, discussed in Sect. III-B.
4) Stop the anytime SAT-based algorithm after 60 sec. Let the best model so far be μ.
5) Re-invoke SATLike for 15 seconds to improve μ, if possible. Go to step 3.

III. Polosat Modifications

We assume that a MaxSAT instance comprises a set of *hard* satisfiable clauses H and a *target bit-vector (target)* $T = \{t_n, t_{n-1}, \ldots, t_1\}$, where each *target bit* t_i is a Boolean variable associated with a strictly positive integer weight w_i. The *weight of a variable assignment* μ is $O(T, \mu) = \sum_{i=1}^{n} \mu(t_i) \times w_i$, that is, the overall weight of T's bits, satisfied by μ. Given a MaxSAT instance, a MaxSAT solver is expected to return a model having the minimum possible weight.

Polosat [8] can be understood as a SAT-based local search algorithm. First, it invokes a SAT solver to get the first model and stashes that model in μ. Then it enters a loop, where each iteration is called an *epoch*. Each epoch tries to improve the best model so far μ. The algorithm finishes and returns μ, when a certain epoch cannot improve μ anymore. In addition, we apply an *adaptive strategy* that stops Polosat forever and falls back to SAT whenever the model generation rate of Polosat is too slow (1 and 2 models per second for the weighted and unweighted components, respectively).

Each epoch goes over the so-called *bad target bits B*, where a target bit is considered *bad* if it has not been assigned 0 in any model from the beginning of the current epoch. The original algorithm in [8] tries to flip each bad target bit t_i by sending the SAT solver the so-called *flip-query* with $\neg t_i$ as an assumption. Note that if the flip-query finds any model, it must be different from every other model encountered during the current epoch, since the current bad target bit is enforced to 0. If a model better than μ is found, μ is updated. The set of the bad target bits *B* is updated, whenever *any* new model is found. In addition, to simulate local search further, `Polosat` applies the `TORC` polarity selection heuristic [6]; see [8] for details.

We modified `Polosat` already in `TT-Open-WBO-Inc-20` as follows. Let t_i be the current bad target bit, encountered by `Polosat`. `TT-Open-WBO-Inc-20` uses an additional SAT query, called the *prefix-query*, prior to the flip-query. For the prefix-query, the SAT solver is provided with the assumption $\neg t_i$ (as in the original `Polosat`) along with a set of assumptions assigning the target bit variables $t_j : 1 \leq j < i$ their polarity in μ. The prefix-query looks for a new model in a more restricted context, induced by the value in μ of the current target prefix. It is expected to come back faster than the flip-query, because of the additional assumptions. If the prefix-query succeeds to improve the best model, the solver skips the flip-query for the current bad target bit. Otherwise, the flip-query is applied as usual.

In our new version of the solver, we modified the queries to the SAT solver further, where the modifications differ for the weighted and unweighted components, respectively.

A. `Polosat` *in the Weighted Component*

The modification is similar to the one we have described above, that is, we apply the prefix-query, followed by the flip-query, except for the following single difference. The flip-query is now skipped whenever the prefix-query finds *any* model (rather than whenever it succeeds to improve the best model so far). The updated algorithm is shown in Fig. 1.

B. `Polosat` *in the Unweighted Component*

Let the *full-query* be a SAT invocation, where the solver is provided with the assumption $\neg t_i$ (as in the original `Polosat`) along with a set of assumptions assigning *all* the target bit variables, but t_i (that is, $t_j : 1 \leq j \neq i \leq n$) their polarity in μ. Note that the search space during the full-query is even more restricted than during the prefix-query.

Our unweighted component sends the solver the flip-query, followed by the full-query, where the full-query is skipped, whenever the flip-query succeeds to improve the best model so far. The algorithm is shown in Fig. 2.

Algorithm 1 `polosat-weighted`

$\mu := \text{SAT}()$ ▷ μ: the best model so far
$g := 1$ ▷ g is 1 iff the current epoch is good
while g **do** ▷ One loop is an epoch
  $B := \{t : t \in T, \mu(t) = 1\}$
  $g := 0$
  while B is not empty **do**
    $t_i := B.front(); B.dequeue()$
    $P := \{t_j \in T : j < i\}$
    $\sigma := \text{SAT}(\{\neg t_i\} \cup \{t : t \in P \wedge \mu(t) = 1\} \cup \{\neg t : t \in P \wedge \mu(t) = 0\})$
    if SAT **then** ▷ Satisfiable
      if $\Psi(\sigma) < \Psi(\mu)$ **then** $\mu := \sigma$ and $g := 1$
      $B := \{t : t \in B, \sigma(t) = 1\}$
    else
      $\sigma := \text{SAT}(\{\neg t_i\})$
      if SAT **then** ▷ Satisfiable
        if $\Psi(\sigma) < \Psi(\mu)$ **then** $\mu := \sigma$ and $g := 1$
        $B := \{t : t \in B, \sigma(t) = 1\}$
return μ

Algorithm 2 `polosat-unweighted`

$\mu := \text{SAT}()$ ▷ μ: the best model so far
$g := 1$ ▷ g is 1 iff the current epoch is good
while g **do** ▷ One loop is an epoch
  $B := \{t : t \in T, \mu(t) = 1\}$
  $g := 0$
  while B is not empty **do**
    $t_i := B.front(); B.dequeue()$
    $\sigma := \text{SAT}(\{\neg t_i\})$
    if SAT **then** ▷ Satisfiable
      if $\Psi(\sigma) < \Psi(\mu)$ **then** $\mu := \sigma$ and $g := 1$
      $B := \{t : t \in B, \sigma(t) = 1\}$
    if not SAT or $\Psi(\sigma) \geq \Psi(\mu))$ **then**
      $P := \{t_j \in T : j \neq i\}$
      $\sigma := \text{SAT}(\{\neg t_i\} \cup \{t : t \in P \wedge \mu(t) = 1\} \cup \{\neg t : t \in P \wedge \mu(t) = 0\})$
      if SAT **then** ▷ Satisfiable
        if $\Psi(\sigma) < \Psi(\mu)$ **then** $\mu := \sigma$ and $g := 1$
        $B := \{t : t \in B, \sigma(t) = 1\}$
return μ

REFERENCES

[1] F. Bacchus, J. Berg, , M. Järvisalo, and R. Martins, editors. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, Department of Computer Science Report Series B 2020-2, Finland, 2020. Department of Computer Science, University of Helsinki.

[2] S. Cai and Z. Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artif. Intell.*, 287:103354, 2020.

[3] S. Joshi, P. Kumar, S. Rao, and R. Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.*, 11(1):73–97, 2019.

[4] Z. Lei and S. Cai. Satlike-c: Solver description. In Bacchus et al. [1].

[5] Z. Lei and S. Cai. Satlike-c(w): Solver description. In Bacchus et al. [1].

[6] A. Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *FMCAD 2019*, pages 193–202.

[7] A. Nadel. Solving MaxSAT with bit-vector optimization. In *SAT 2018*, pages 54–72, 2018.

[8] A. Nadel. On optimizing a generic function in SAT. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 205–213. IEEE, 2020.

[9] A. Nadel. Polarity and variable selection heuristics for SAT-based anytime MaxSAT. *J. Satisf. Boolean Model. Comput.*, 12(1):17–22, 2020.

[10] A. Nadel. TT-Open-WBO-Inc-20: an Anytime MaxSAT Solver Entering MSE'20. In Bacchus et al. [1].

SOLVERS FROM PREVIOUS EVALUATIONS

Loandra in the 2020 MaxSAT Evaluation

Jeremias Berg*, Emir Demirović†, Peter Stuckey‡
*HIIT, Department of Computer Science, University of Helsinki, Finland
†Delft University of Technology, The Netherlands
‡Monash University, Australia

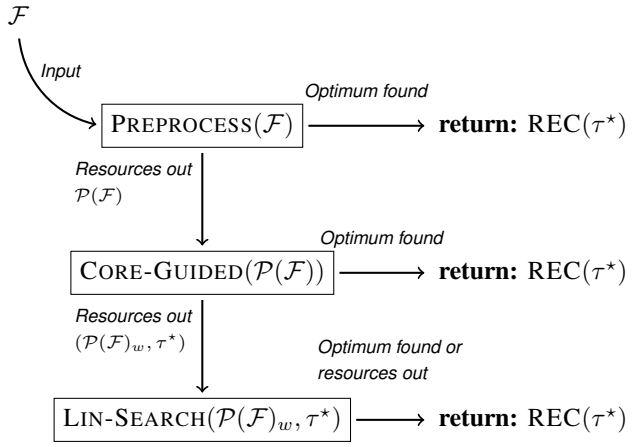

Fig. 1: The structure of Loandra.

I. PRELIMINARIES

We briefly overview the Loandra MaxSAT-solver as it participated in the incomplete track of the 2020 MaxSAT Evaluation, focusing especially on the differences between the 2019 and 2020 versions, more detailed descriptions can be found in [4], [10]. Loandra owes much of its existence to Open-WBO [11], we thank the developers of Open-WBO for their work .

We assume familiarity with conjunctive normal form (CNF) formulas and weighted partial maximum satisfiability (MaxSAT). Treating a CNF formula as a set of clauses a MaxSAT instance $\mathcal{F}$ consists of two CNF formulas, the hard clauses F_h and the soft clauses F_s, as well a weight w_c associated with each $C \in F_s$. A solution to $\mathcal{F}$ is an assignment τ that satisfies F_h. The cost of a solution τ is the sum of weights of the soft clauses falsified by τ. An optimal solution is one with minimum cost over all solutions. An unsatisfiable core κ of $\mathcal{F}$ is a subset of soft clauses s.t. $F_h \wedge \kappa$ is unsatisfiable.

II. STRUCTURE OF LOANDRA

Figure 1 overviews the structure of Loandra. In short, Loandra implements core-boosted linear search [4] augmented with tightly integrated MaxSAT preprocessing [3], [9], [10], [2]. More specifically, Loandra consists of three main components: a) Preprocessing, b) Core-guided search, c) Linear search.

a) Preprocessing: On input $\mathcal{F}$, the execution starts by invoking the MaxPre [9] preprocessor on $\mathcal{F}$ using the standard techniques of MaxPre *except for blocked clause elimination (BCE)*. The reason we are not using BCE is that, as detailed in [10], intermediate solutions to instances preprocessed with BCE are more prone to having their costs miss-interpreted by the core-guided and linear search components of Loandra. If MaxPre can not compute the optimal solution to $\mathcal{F}$, the preprocessed instance $\mathcal{P}(\mathcal{F})$ is handed to the core guided phase (CORE-GUIDED in Figure 1), reusing the assumption variables introduced during preprocessing [3].

b) Core-guided search: The core-guided phase is unchanged from the 2019 version; as the instantiation of the core-guided algorithm, we use a reimplementation of PMRES [12] extended with weight aware core extraction (WCE) [5] and clause hardening. If CORE-GUIDED is able to find an optimal solution τ to $\mathcal{P}(\mathcal{F})$, an optimal solution $\text{REC}(\tau)$ to $\mathcal{F}$ is reconstructed and returned. Otherwise i.e. if the core-guided phase runs out of time, the final working instance $\mathcal{P}(\mathcal{F})_w$ and τ^*, the best found solution to it is handed to the linear search component LIN-SEARCH.

c) Linear search: LIN-SEARCH, the linear search phase of Loandra is an implementation of the SAT/UNSAT linear search algorithm [6], extended with solution guided phase saving and varying resolution in the style of LinSBPS [7]. The component is for the most part the same as in the 2019 version. The main difference is, that in the start of each resolution, the currently best known solution τ^* is minimized in order to alleviate the missinterpretation of costs that might happen due to preprocessing in the context of incomplete solving [10].

More specifically, let $\mathcal{P}(\mathcal{F})_w = \mathcal{P}(\mathcal{F}) \cup \text{CARD}$ be the working instance of LIN-SEARCH where $\mathcal{P}(\mathcal{F})$ is the preprocessed instance computed by MaxPre and CARD are the constraints added in the core-guided phase. At the start of each resolution, LIN-SEARCH computes a set $\mathcal{B}_s$ of blocking variables and an upper bound UB over which the PB constraint is built. The upper bound is computed based on τ^*, the current best known solution to $\mathcal{P}(\mathcal{F})_w$. However, as shown in [10], there can be a significant difference between $\text{COST}(\mathcal{P}(\mathcal{F}), \tau^*)$, the cost of τ^* w.r.t to $\mathcal{P}(\mathcal{F})$, and $\text{COST}(\mathcal{F}, \text{REC}(\tau^*))$, the cost of the solution to $\mathcal{F}$ reconstructed from τ^*. This difference might result UB, and as consequence the whole PB constraint, being much larger than actually required. In order to alleviate this issue LIN-SEARCH uses a simple, procedure that iteratively fixes all variables in $\mathcal{B}_s$ in the following manner. In each iteration, all variables in $\mathcal{P}(\mathcal{F})$

are fixed to the polarities that they are assigned to by τ^*. Additionally an unfixed variable $b \in \mathcal{B}_s$ is fixed to false (i.e. to not incur cost). Then the SAT solver is used to extend these fixings into a satisfying assignment of $\mathcal{P}(\mathcal{F})_w$. If such an assignment τ_b^* can be found, that assignment will have $\text{COST}(\mathcal{P}(\mathcal{F}), \tau_b^*) < \text{COST}(\mathcal{P}(\mathcal{F}), \tau^*)$ while also agreeing with τ^* on all variables in $\mathcal{P}(\mathcal{F})$. The variable b is then fixed to false in subsequent iterations. Otherwise, the variable b is fixed to true in subsequent iterations. Notice that, due to the nature of constraints added by CORE-GUIDED, each individual SAT-solver call is solvable by unit propagation alone (the constraints in CARD are basically cardinality constraints). Even so, preliminary experiments showed that the minimization procedure is too expensive to run for each new solution found, which is why we restrict it to once per resolution.

The linear phase runs until either finding an optimal solution, or running out of time, at which point a reconstruction $\text{REC}(\tau^*)$ of the currently best known solution τ^* to $\mathcal{P}(\mathcal{F})_w$ is returned. Notice that the reconstruction of a solution happens only once, we use the standard, linear time, reconstruction algorithm as implemented by MaxPre.

III. IMPLEMENTATION DETAILS

All algorithms are implemented on top of the publicly available Open-WBO system [11] using Glucose 4.1 [1] as the back-end SAT solver. In order to minimize I/O overhead, we make direct use of the preprocessor interface offered by MaxPre. The linear search algorithm uses the generalized totalizer encoding [8] to convert the PB constraints needed in linear search to CNF. In the evaluation, we set a 30s time limit for the preprocessing phase and a 30 second time limit for the core-guided phase. These limits were chosen based on preliminary experiments. On weighted instances, the core-guided phase is also terminated when the stratification bound would be lowered to 1. On unweighted instances the phase is terminated at the latest after extracting one set of disjoint cores.

IV. COMPILATION AND USAGE

Building and using Loandra resembles building and using Open-WBO. Before building loandra, the maxpre library needs to be built by invoking MAKE LIB in the maxpre subfolder. Afterwards, a statically linked version of Loandra in release mode can be built by running MAKE RS in the base folder.

After building, Loandra can be invoked from the terminal. Except for the formula file, Loandra accepts a number of command line arguments: the flag *-pmreslin-cglim* sets the maximum time that the core-guided phase can run for (in seconds). The rest of the flags resemble the flags accepted by Open-WBO; invoke ./loandra_static –help-verb for more information.

REFERENCES

[1] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proc IJCAI*. Morgan Kaufmann Publishers Inc., 2009, pp. 399–404.

[2] A. Belov, A. Morgado, and J. Marques-Silva, "SAT-based preprocessing for MaxSAT," in *Proc. LPAR-19*, ser. Lecture Notes in Computer Science, vol. 8312. Springer, 2013, pp. 96–111.

[3] J. Berg, P. Saikko, and M. Järvisalo, "Improving the effectiveness of SAT-based preprocessing for MaxSAT," in *Proc. IJCAI*. AAAI Press, 2015, pp. 239–245.

[4] J. Berg, E. Demirovic, and P. J. Stuckey, "Core-boosted linear search for incomplete maxsat," in *CPAIOR*, ser. Lecture Notes in Computer Science, vol. 11494. Springer, 2019, pp. 39–56.

[5] J. Berg and M. Järvisalo, "Weight-aware core extraction in SAT-based MaxSAT solving," in *Proc. CP*, ser. Lecture Notes in Computer Science, 2017, to appear.

[6] D. L. Berre and A. Parrain, "The sat4j library, release 2.2," *J. Satisf. Boolean Model. Comput.*, vol. 7, no. 2-3, pp. 59–6, 2010. [Online]. Available: https://satassociation.org/jsat/index.php/jsat/article/view/82

[7] E. Demirovic and P. J. Stuckey, "Techniques inspired by local search for incomplete maxsat and the linear algorithm: Varying resolution and solution-guided search," in *CP*, ser. Lecture Notes in Computer Science, vol. 11802. Springer, 2019, pp. 177–194.

[8] S. Joshi, R. Martins, and V. M. Manquinho, "Generalized totalizer encoding for pseudo-boolean constraints," in *Proc. CP*, ser. LNCS, vol. 9255, 2015, pp. 200–209.

[9] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "Maxpre: An extended maxsat preprocessor," in *SAT*, ser. Lecture Notes in Computer Science, vol. 10491. Springer, 2017, pp. 449–456.

[10] M. Leivo, J. Berg, and M. Järvisalo, "Preprocessing in incomplete maxsat solving," in *Proc ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. ???? IOS Press, 2020, p. (to appear).

[11] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: A modular MaxSAT solver," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 8561. Springer, 2014, pp. 438–445.

[12] N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided MaxSAT resolution," in *Proc. AAAI*. AAAI Press, 2014, pp. 2717–2723.

Open-WBO-Inc in MaxSAT Evaluation 2020

Saurabh Joshi, Prateek Kumar
{sbjoshi,cs15btech11031}@iith.ac.in
Indian Institute of Technology Hyderabad, India

Sukrut Rao
sukrut.rao@mpi-inf.mpg.de
Max Planck Institute for Informatics, Germany

Ruben Martins
rubenm@cs.cmu.edu
CMU, USA

I. Introduction

Open-WBO-Inc [1], [2] is developed on top of Open-WBO [3], [4], [5] and placed first and second on the weighted incomplete tracks for 60 and 300 seconds respectively in the MaxSAT Evaluation 2018, and third on both these tracks in the MaxSAT Evaluation 2019. For many applications that can be encoded into MaxSAT, it is important to quickly find solutions even though these may not be optimal. Open-WBO-Inc is designed to find a good solution[1] in a short amount of time. Since Open-WBO-Inc is based on Open-WBO, it can use any MiniSAT-like solver [6]. For this evaluation, we use Glucose 4.1 [7] as our back-end SAT solver.

II. Algorithms

For the MaxSAT Evaluation 2020, we restrict Open-WBO-Inc to the weighted category where it uses the novel approximation algorithms that have been recently proposed [1], [2]. In particular, we submitted three versions of Open-WBO-Inc: inc-bmo-complete, inc-bmo-satlike, and inc-bmo-satlike19.

All versions are based on bounded multilevel optimization [8] using a variant of linear search algorithm SAT-UNSAT [9]. The algorithms used in these versions consider n objective functions where n is the number of different weights in the MaxSAT instance. This is done by performing a sequence of calls to a SAT solver and refining an upper bound μ on the number of unsatisfied soft clauses. To restrict μ at each iteration, we need to encode cardinality constraints into CNF, for which incremental Totalizer encoding [4] has been used. Once the upper bound μ for a given objective function cannot be improved, it is frozen, and the next objective function in the order is optimized.

An optimal solution, if found when using this algorithm, is not necessarily an optimal solution for the input formula. inc-bmo-complete and inc-bmo-satlike versions differ between them when this occurs. inc-bmo-complete keeps the best-known solution and resumes the search using the LSU algorithm which can potentially find better solutions and prove optimality. In contrast, inc-bmo-satlike changes the search algorithm to SATLike [10], a MaxSAT stochastic algorithm. The best model found by the first phase is passed to SATLike as its initial starting model.

[1]By "good solution" we mean that it can be potentially suboptimal but is not far from the optimal solution.

The inc-bmo-satlike19 version corresponds to the best performing version of Open-WBO-Inc in the MaxSAT Evaluation 2019. For the versions of this year, we added a conflict limit of 10^7 on each SAT call when performing the multilevel optimization phase. This prevents the solver from being stuck in some optimization level and never entering the final phase. We have also included the Target-Optimum-Rest-Conservative (TORC) and Target-Score-Bum (TSB) heuristics [11]. The TORC heuristic changes the default polarity of the SAT solver to take into consideration the MaxSAT formula. Relaxation variables that may appear in the cardinality constraints of the multilevel optimization algorithm are always set to polarity *false*. For the remaining variables, the polarity is set according to the best model found during search. The TSB heuristic bumps the score of all relaxation variables to make them more likely to be picked at the beginning of the search. Additionally, we also now support printing a compact certificate using 0's and 1's instead of variable ids.

III. Availability

We submit the source of Open-WBO-Inc as part of our submissions to the MaxSAT Evaluations 2020. The inc-bmo-complete version and the full Open-WBO-Inc framework is available under a MIT license in GitHub at *https://github.com/sbjoshi/Open-WBO-Inc*.

Acknowledgments

We would like to thank Laurent Simon and Gilles Audemard for allowing us to use Glucose in the MaxSAT Evaluation. We would also like to thank Vasco Manquinho, Inês Lynce, Mikoláš Janota, Miguel Terra-Neves and Norbert Manthey for their contributions to Open-WBO on which Open-WBO-Inc is based.

References

[1] S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation Strategies for Incomplete MaxSAT," in *CP*. Springer, 2018.
[2] S. Joshi, P. Kumar, S. Rao, and R. Martins, "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT," in *Journal on Satisfiability, Boolean Modeling and Computation*. IOS Press, 2019.
[3] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.
[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.
[5] M. Neves, R. Martins, M. Janota, I. Lynce, and V. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.
[6] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.

[7] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.
[8] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.
[9] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 59–6, 2010.
[10] Z. Lei and S. Cai, "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT," in *IJCAI*. ijcai.org, 2018, pp. 1346–1352.
[11] A. Nadel, "Anytime weighted maxsat with improved polarity selection and bit-vector optimization," in *Proc. Formal Methods in Computer Aided Design*, C. W. Barrett and J. Yang, Eds. IEEE, 2019, pp. 193–202.

Pacose: An Iterative SAT-based MaxSAT Solver

Tobias Paxian, Bernd Becker
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 051
79110 Freiburg, Germany
{ paxiant | becker }@informatik.uni-freiburg.de

I. Overview

Pacose is a SAT-based MaxSAT solver, using two incremental CNF encodings, a binary adder [1] and the Dynamic Polynomial Watchdog (DPW) [2], for Pseudo-Boolean (PB) constraints. It is an extension of QMaxSAT 2017 [3], based on Glucose 4.2.1 [4] SAT solver. It uses a Boolean Multilevel Optimization (BMO) pre- / inprocessing method to simplify the instances. Additionally a trimming method is applied to cut off unsatisfiable soft clauses and find a good initial satisfiable weight to reduce the size of the encoding.

II. Pre- / Inprocessing

The 2019 version of Pacose contains two new pre-/inprocessing methods, Generalized Boolean Multilevel Optimization (GBMO) and a trimming algorithm.

Multi-Objective Combinatorial Optimization (MOCO) [5] problems are addressing multiple optimization problems with possibly conflicting purposes. Boolean Multilevel Optimization (BMO) [6], [7] is the mapping of MOCO to MaxSAT solving. We generalized the plain variant of Boolean Multilevel Optimization thereby making it possible to split additional instances, even in cases where the weight differences of the sum of smaller weights is non-strictly smaller than the next biggest weight.

The trimming algorithm tries to satisfy each soft clause at least once with the additional goal to find a good approximation of the weight. It works in two phases, in the first phase it optimizes the overall weight and in the second phase it satisfies as many soft clauses as possible in the next solver call. After a timeout which is based on the number of soft clauses, it switches from the first phase to the second. An additional timeout for each incrementally solver call is included.

III. Encoding and Algorithm

Our DPW encoding is based on the Polynomial Watchdog (PW) encoding [8], which uses totalizer networks [9]. Essentially the DPW encoding employs multiple totalizer networks to perform a binary addition with carry on the sorted outputs. A special algorithm to solve these instances incremental is presented in [2].

Additionally the adder network [1] is used which has a linear complexity in encoding size in contrast to at least $\mathcal{O}(n^2)$ for the DPW sorting network. With the adder network many complementary instances to the DPW encoding can be solved and therefore it is well suited, to be chosen, together with DPW by a heuristic, as described in the following chapter. The algorithm and encoding are partly adapted and inspired from QMaxSAT.

This work is supported by DFG "Algebraic Fault Attacks" (BE 1176/20-2)

IV. Heuristics

Pacose uses straightforward heuristics based on available MaxSAT benchmarks. All heuristics are based on the number of soft clauses and the overall sum of soft weights.

- *Encoding:* The DPW encoding empirically works best if the average weight for soft clauses is small, or the overall sum of soft weights is huge (bigger than 80 billion). For the other benchmarks the binary adder is chosen.
- *Trimming:* As for instances with only a few soft clauses the trimming preprocessing algorithm is not effective, it is only used if the benchmark contains at least a certain amount of soft clauses.
- *Compression Rate:* For benchmarks with only a few soft clauses, the encoding is smaller and additional clauses can be added. Therefore the binary adder encoding can solve overall more benchmarks if the compression rate is chosen accordingly.

References

[1] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.
[2] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted MaxSAT," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2018, pp. 37–53.
[3] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, pp. 95–100, 2012.
[4] G. Audemard and L. Simon, "On the glucose SAT solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.
[5] E. L. Ulungu and J. Teghem, "Multi-objective combinatorial optimization problems: A survey," *Journal of Multi-Criteria Decision Analysis*, vol. 3, no. 2, pp. 83–104, 1994.
[6] J. Argelich, I. Lynce, and J. Marques-Silva, "On solving boolean multilevel optimization problems," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.
[7] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.
[8] O. Bailleux, Y. Boufkhad, and O. Roussel, "New encodings of pseudo-boolean constraints into CNF," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2009, pp. 181–194.
[9] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in *Principles and Practice of Constraint Programming–CP 2003*. Springer, 2003, pp. 108–122.

Stable Resolving

Julian Reisch
Synoptics GmbH Dresden, Germany
julian.reisch@synoptics.de

Peter Großmann
Synoptics GmbH Dresden, Germany
peter.grossmann@synoptics.de

***Abstract*—We describe Stable Resolving (SR) [6], a solver competing in the incomplete track of the 2020 MaxSAT evaluation. SR is a randomized local search heuristic for both weighted and unweighted instances. The algorithm consists of three steps that are executed repeatedly. In a perturbation, the search space is explored. Then, local improvements are performed by flipping the signs of variables in over-satisfied clauses. Finally, a simulated annealing solution checking allows for leaving local optima.**

***Index Terms*—MaxSAT, heuristic, local search, incomplete solving**

I. Overall Procedure

The solver starts with a SAT-based preprocessing and a call of the SAT-solver Glucose [2] for an initial solution before the three steps of perturbation, improvements and solution checking are executed repeatedly until the global timeout is reached. The procedure is shown in Algorithm 1.

Algorithm 1 StableResolving()

```
Preprocess()
CalculateInitialSolution()
while timeout has not been reached do
    Perturbation()
    Improvements()
    SolutionChecking()
end
```

This algorithm structure has been proposed by [1] for the maximum independent set problem and applied in an adapted form to transformed MaxSAT instances by [7].

II. Preprocess

The preprocessing consists of repeated unit clause and pure literal propagation and bounded variable elimination (cf. e.g. [4]) until no more pure literals or unit clauses are can be propagated or a sixth of the global timeout is exceeded. Since these operations are only sound for hard clauses, we label the soft clauses, consider them as hard clauses and add an extra soft clause for each label (cf. [3] for details). Then, for an initial solution the SAT solver glucose [2] is called.

III. Perturbation

In the perturbation, we aim at altering the solution in order to explore the search space even though the solution might worsen. More precisely, we flip the sign of a variable picked uniformly at random in unsatisfied clauses, selected uniformly at random in the set of unsatisfied clauses. We do not pick clauses or variables twice in this procedure. In addition, in every n-th perturbation call, the selected unsatisfied clauses are considered hard clauses and given to the SAT-solver glucose that forces a solution where these clauses are satisfied. Glucose has a time limit of 5 seconds for this. A subset of unsatisfied clauses is found by first sampling a random number k from the geometric distribution with parameter p_1 and then selecting k clauses from the set of unsatisfied clauses. As most of the formula's clauses remain satisfied during many iterations, we keep and update a superset of the unsatisfied clauses during the whole algorithm to speed up the sampling from this superset instead collecting all unsatisfied clauses in each perturbation step. Clauses that become unsatisfied in the perturbation step are added to a set of unsatisfied *candidate* clauses. Finally, the perturbation has a plateau search where, again, a random number k is sampled from the geometric distribution with parameter p_2. Then, k variables are picked uniformly at random and their signs are flipped if no clause becomes unsatisfied by the flip.

IV. Improvements

The improvement part is the core of SR and works in the following way. Starting from a random (unsatisfied) candidate clause, SR flips the sign of a randomly chosen variable from a set of currently unsatisfied clauses. If this flip causes other clauses to become unsatisfied, they are added to this set. If it leads to an over-satisfaction of a clause, the algorithm attempts to flip yet another of this over-satisfied clause's variables' signs if no further clauses become unsatisfied by that second flip. We keep a vector containing the number of true literals for each clause, denoted the clause's *stability*, for this step and perform it whenever the stability of a clause grows from 1 to 2. Moreover, we do not flip a variable's sign twice during a single improvement step. After no further clauses in the set of currently unsatisfied clauses can be satisfied, or an iteration limit of l is reached, the improvement for the clause ends and we carry on with the next candidate clause. After all candidate clauses have been tried to improve, the improvement part ends.

V. Solution Checking

If we see the best solution found so far, we save it and continue. However, it is possible that the improvements could not compensate the solution's worsening of the perturbation step. Nevertheless, with a probability that decreases exponentially in the time elapsed, we accept that worse solution because it might help to leave local optima. This technique is known as

Flag	Description	Type
-z	global timeout	int
-o	file path the solution output	string
-innermaxit	l	int
-maxstepsworse	m	int
-geom	p_1	int
-plateau	p_2	int
-maxstepsperturbtosat	n	int

simulated annealing [5]. However, after m iterations without an update of the best solution, we restore the best solution.

VI. Implementation and Usage

The implementation of SR is in C++. The various parameters within the algorithm are set at the beginning and according to selected features of the instance at hand, such as the number of soft clauses or their average weight. If desired, the parameters can be set manually when calling the solver with the flag *-autoparam*. This disables the automatic setting of parameters and all parameters listed in Table VI can be set.

References

[1] D. V. Andrade, M. G. C. Resende, R. F. F. Werneck, "Fast local search for the maximum independent set problem", in J. Heuristics, 18(4), 2012

[2] G. Audemard, L. Simon, "Predicting learnt clauses quality in modern sat solvers", in Proceedings of the 21st International Jont Conference on Artifical Intelligence (IJCAI'09), San Francisco, CA, USA, 2009

[3] A. Belov, A. Morgado, J. Marques-Silva, "Sat-based preprocessing for maxsat", in McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning. pp. 96–111, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013

[4] M. Davis, H. Putnam, "A computing procedure for quantification theory", in J. ACM7(3), pp. 201–215, 1960

[5] M. Pincus, "A Monte-Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems", in Operation Research, 18(6), 1970

[6] J. Reisch, P. Großmann, N. Kliewer, "Stable Resolving - A Randomized Local Search Heuristic for MaxSAT", unpublished paper under review, 2020

[7] J. Reisch, P. Großmann, N. Kliewer, "Conflict Resolving - A Maximum Independent Set Heuristics for Solving MaxSAT", in Proceedings of the 22nd International Multiconference Information Society, 1, pp. 67-71, 2019

BENCHMARKS

Planning with Learned Binarized Neural Networks Benchmarks for MaxSAT Evaluation 2021

Buser Say
Monash University
Melbourne, Australia
buser.say@monash.edu

Scott Sanner
University of Toronto
Toronto, Canada
ssanner@mie.utoronto.ca

Jo Devriendt
KU Leuven
Leuven, Belgium
jo.devriendt@kuleuven.be

Jakob Nordström
University of Copenhagen
Copenhagen, Denmark
jn@di.ku.dk

Peter J. Stuckey
Monash University
Melbourne, Australia
peter.stuckey@monash.edu

***Abstract*—This document provides a brief introduction to learned automated planning problem where the state transition function is in the form of a binarized neural network (BNN), presents a general MaxSAT encoding for this problem, and describes the four domains, namely: Navigation, Inventory Control, System Administrator and Cellda, that are submitted as benchmarks for MaxSAT Evaluation 2021.**

***Index Terms*—binarized neural networks, automated planning**

I. Introduction

Automated planning studies the reasoning side of acting in Artificial Intelligence, and automates the selection and ordering of actions to reach desired states of the world as best as possible [1]. An automated planning problem represents dynamics of the real-world using a model, which can either be manually encoded [2]–[6], or learned from data [7]–[10]. In this document, we focus on the latter.

Automated planning with deep neural network (DNN) learned state transition functions is a two stage data-driven framework for learning and solving automated planning problems with unknown state transition functions [11]–[13]. The first stage of the framework learns the unknown state transition function from data as a DNN. The second stage of the framework plans optimally with respect to the learned DNN by solving an equivalent optimization problem (e.g., a mixed-integer programming model [11], [14]–[16], a 0–1 integer programming model [12], [17], a weighted partial MaxSAT model [12], [17], a constraint programming model [18], or a pseudo-Boolean optimization model [18]). In this document, we focus on the second stage of the data-driven framework where the learned DNN is a binarized neural network (BNN) [19].

The remaining of the document is organized as follows. We begin with the description of the learned automated planning problem and the binarized neural network (BNN). Then we present the weighted partial MaxSAT model of the general learned automated planning problem, and conclude with the description of four learned automated planning domains, namely: Navigation, Inventory Control, System Administrator and Cellda, that are submitted as benchmarks for MaxSAT Evaluation 2021.

II. Automated Planning with Learned Binarized Neural Network State Transitions

A. Problem Definition

A fixed-horizon learned deterministic automated planning problem [11], [12], [18] is a tuple $\tilde{\Pi} = \langle S, A, C, \tilde{T}, V, G, R, H \rangle$, where $S = \{s_1, \ldots, s_n\}$ and $A = \{a_1, \ldots, a_m\}$ are sets of state and action variables for positive integers $n, m \in \mathbb{Z}^+$ with domains $D_{s_1}, \ldots, D_{s_n}$ and $D_{a_1}, \ldots, D_{a_m}$ respectively. Moreover, $C : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \rightarrow \{true, false\}$ is the global function, $\tilde{T} : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \rightarrow D_{s_1} \times \cdots \times D_{s_n}$ is the learned state transition function, and $R : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \rightarrow \mathbb{R}$ is the reward function. Finally, V is a tuple of constants $\langle V_1, \ldots, V_n \rangle \in D_{s_1} \times \cdots \times D_{s_n}$ denoting the initial values of all state variables, $G : D_{s_1} \times \cdots \times D_{s_n} \rightarrow \{true, false\}$ is the goal state function, and $H \in \mathbb{Z}^+$ is the planning horizon.

A *solution* to (i.e., a *plan* for) $\tilde{\Pi}$ is a tuple of values $\bar{A}^t = \langle \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle \in D_{a_1} \times \cdots \times D_{a_m}$ for all action variables A over time steps $t \in \{1, \ldots, H\}$ such that $\tilde{T}(\langle \bar{s}_1^t, \ldots, \bar{s}_n^t, \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle) = \langle \bar{s}_1^{t+1}, \ldots, \bar{s}_n^{t+1} \rangle$ and $C(\langle \bar{s}_1^t, \ldots, \bar{s}_n^t, \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle) = true$ holds for time steps $t \in \{1, \ldots, H\}$, $V_i = \bar{s}_i^1$ for all $s_i \in S$ and $G(\langle \bar{s}_1^{H+1}, \ldots, \bar{s}_n^{H+1} \rangle) = true$. It has been shown that finding a feasible solution to $\tilde{\Pi}$ is *NP*-complete [18]. An *optimal solution* to (i.e., an *optimal plan* for) $\tilde{\Pi}$ is a solution such that the total reward $\sum_{t=1}^{H} R(\langle \bar{s}_1^{t+1}, \ldots, \bar{s}_n^{t+1}, \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle)$ is maximized.

We assume that the domains of action and state variables are binary unless otherwise stated[1], the functions C, G, R and

[1]When the domain of a variable is not binary (e.g., see Inventory Control), we can use the following approximation $x \approx (-2^{m_1-1} x_{m_1} + \sum_{i=1}^{m_1-1} 2^{i-1} x_i) 10^{m_2}$ for integers $m_1 \in \mathbb{Z}^+$ and $m_2 \in \mathbb{Z}$.

function $\tilde{T}$ are known, functions C, G can be equivalently represented by $J_C \in \mathbb{Z}^+$ and $J_G \in \mathbb{Z}^+$ linear constraints, function R is a linear expression and function $\tilde{T}$ is a learned BNN [19].

B. Binarized Neural Networks

Binarized neural networks (BNNs) are DNNs with binarized weights and activation functions [19]. Given L layers with layer width W_l of layer $l \in \{1, \ldots, L\}$, and a set of neurons $J(l) = \{u_{1,l}, \ldots, u_{W_l,l}\}$, is stacked in the following order.

a) Input Layer: The first layer consists of neurons $u_{i,1} \in J(1)$ that represent the domain of the learned state transition function $\tilde{T}$ where neurons $u_{1,1}, \ldots, u_{n,1} \in J(1)$ represent the state variables S and neurons $u_{n+1,1}, \ldots, u_{n+m,1} \in J(1)$ represent the action variables A. During the training of the BNN, values 0 and 1 of action and state variables are represented by -1 and 1, respectively.

b) Batch Normalization Layers: For layers $l \in \{2, \ldots, L\}$, Batch Normalization [20] sets the weighted sum of outputs at layer $l-1$ in $\triangle_{j,l} = \sum_{i \in J(l-1)} w_{i,j,l} y_{i,l-1}$ to inputs $x_{j,l}$ of neurons $u_{j,l} \in J(l)$ using the formula $x_{j,l} = \frac{\triangle_{j,l} - \mu_{j,l}}{\sqrt{\sigma^2_{j,l} + \epsilon_{j,l}}} \gamma_{j,l} + \beta_{j,l}$, where $y_{i,l-1}$ is the output of neuron $u_{i,l-1} \in J(l-1)$, and the parameters are the weight $w_{i,j,l}$, input mean $\mu_{j,l}$, input variance $\sigma^2_{j,l}$, numerical stability constant $\epsilon_{j,l}$, input scaling $\gamma_{j,l}$, and input bias $\beta_{j,l}$, all computed at training time.

c) Activation Layers: Given input $x_{j,l}$, the activation function $y_{j,l}$ computes the output of neuron $u_{j,l} \in J(l)$ at layer $l \in \{2, \ldots, L\}$, which is 1 if $x_{j,l} \geqslant 0$ and -1 otherwise. The last activation layer consists of neurons $u_{i,L} \in J(L)$ that represent the codomain of the learned state transition function $\tilde{T}$ such that $u_{1,L}, \ldots, u_{n,L} \in J(L)$ represent the state variables S.

The BNN is trained to learn the function $\tilde{T}$ from data that consists of measurements on the domain and codomain of the *unknown* state transition function $T : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \to D_{s_1} \times \cdots \times D_{s_n}$.

III. The Weighted Partial MaxSAT Model

In this section, we present the weighted partial MaxSAT model [12], [17] of the learned automated planning problem.

A. Decision Variables

The weighted partial MaxSAT model uses the following decision variables:

- $X_{i,t}$ encodes whether action variable $a_i \in A$ is executed at time step $t \in \{1, \ldots, H\}$ or not.
- $Y_{i,t}$ encodes whether state variable $s_i \in S$ is true at time step $t \in \{1, \ldots, H+1\}$ or not.
- $Z_{i,l,t}$ encodes whether neuron $u_{i,l} \in J(l)$ in layer $l \in \{1, \ldots, L\}$ is active at time step $t \in \{1, \ldots, H\}$ or not.

B. Parameters

The weighted partial MaxSAT model uses the following parameters:

- $\bar{w}_{i,j,l}$ is the value of the learned BNN weight between neuron $u_{i,l-1} \in J(l-1)$ and neuron $u_{j,l} \in J(l)$ in layer $l \in \{2, \ldots, L\}$.
- $B(j,l)$ is the value of the bias for neuron $u_{j,l} \in J(l)$ in layer $l \in \{2, \ldots, L\}$. Given the values of learned parameters $\bar{\mu}_{j,l}$, $\bar{\sigma}^2_{j,l}$, $\bar{\epsilon}_{j,l}$, $\bar{\gamma}_{j,l}$ and $\bar{\beta}_{j,l}$, the bias is computed as $B(j,l) = \left\lceil \frac{\bar{\beta}_{j,k}\sqrt{\bar{\sigma}^2_{j,l} + \bar{\epsilon}_{j,l}}}{\bar{\gamma}_{j,l}} - \bar{\mu}_{j,l} \right\rceil$.
- $r^s_i \in \mathbb{R}$ and $r^a_i \in \mathbb{R}$ are constants of the reward function R that is in the form of $\sum_{i=1}^{n} r^s_i s_i + \sum_{i=1}^{m} r^a_i a_i$.
- $c^s_{i,j} \in \mathbb{Z}$, $c^a_{i,j} \in \mathbb{Z}$ and $c^k_j \in \mathbb{Z}$ are constants of the set of linear constraints that represent the global function C where each linear constraint $j \in \{1, \ldots, J_C\}$ is in the form of $\sum_{i=1}^{n} c^s_i s_i + \sum_{i=1}^{m} c^a_i a_i \leqslant c^k_j$.
- $g^s_{i,j} \in \mathbb{Z}$ and $g^k_j \in \mathbb{Z}$ are constants of the set of linear constraints that represent the goal state function G where each linear constraint $j \in \{1, \ldots, J_G\}$ is in the form of $\sum_{i=1}^{n} g^s_i s_i \leqslant g^k_j$.

C. Hard Clauses

The weighted partial MaxSAT model uses the following hard clauses.

a) Initial State Clauses: The following conjunction of hard clauses sets the initial value V_i of each state variable $s_i \in S$.

$$\bigwedge_{i=1}^{n} (\neg Y_{i,1} \vee V_i) \wedge (Y_{i,1} \vee \neg V_i) \tag{1}$$

b) Goal State Clauses: The following conjunction of hard clauses encodes the set of linear constraints that represent the goal state function G.

$$\bigwedge_{j=1}^{J_G} Card(\sum_{i=1}^{n} g^s_i Y_{i,H+1} \leqslant g^k_j) \tag{2}$$

In the above notation, $Card$ produces the CNF encoding of a given linear constraint [21].

c) Global Clauses: The following conjunction of hard clauses encodes the set of linear constraints that represent the global function C.

$$\bigwedge_{j=1}^{J_C} \bigwedge_{t=1}^{H} Card(\sum_{i=1}^{n} c^s_i Y_{i,t} + \sum_{i=1}^{m} c^a_i X_{i,t} \leqslant c^k_j) \tag{3}$$

d) BNN Clauses: The following conjunction of hard clauses maps the input and the output of the BNN onto the

TABLE I
THE BNN ARCHITECTURES OF ALL FOUR LEARNED AUTOMATED PLANNING PROBLEMS.

Problem	BNN Structure
Discrete Navigation ($N = 3$)	13:36:36:9
Discrete Navigation ($N = 4$)	20:96:96:16
Discrete Navigation ($N = 5$)	29:128:128:25
Inventory Control ($N = 2$)	7:96:96:5
Inventory Control ($N = 4$)	8:128:128:5
System Administrator ($N = 4$)	16:128:128:12
System Administrator ($N = 5$)	20:128:128:128:15
Cellda (policy=x-axis)	12:256:256:4
Cellda (policy=y-axis)	12:256:256:4

state and action variables.

$$\bigwedge_{i=1}^{n}\bigwedge_{t=1}^{H}(\neg Y_{i,t} \vee Z_{i,1,t}) \wedge (Y_{i,t} \vee \neg Z_{i,1,t}) \tag{4}$$

$$\bigwedge_{i=1}^{m}\bigwedge_{t=1}^{H}(\neg X_{i,t} \vee Z_{i+n,1,t}) \wedge (X_{i,t} \vee \neg Z_{i+n,1,t}) \tag{5}$$

$$\bigwedge_{i=1}^{n}\bigwedge_{t=1}^{H}(\neg Y_{i,t+1} \vee Z_{i,L,t}) \wedge (Y_{i,t+1} \vee \neg Z_{i,L,t}) \tag{6}$$

Finally, the following conjunction of hard clauses encodes the activation function of each neuron in the learned BNN.

$$\bigwedge_{l=2}^{L}\bigwedge_{u_{j,l}\in J(l)}\bigwedge_{t=1}^{H} Act\Big(\Big(\sum_{u_{i,l-1}\in J(l-1)} \bar{w}_{i,j,l}(2Z_{i,l-1,t}-1) + B(j,l) \geqslant 0\Big) = Z_{j,l,t}\Big) \tag{7}$$

In the above notation, Act produces the CNF encoding of a given biconditional constraint [17] by extending the CNF encoding of Cardinality Networks [22].

D. Soft Clauses

The weighted partial MaxSAT model uses the following soft clauses.

a) Reward Clauses: The following conjunction of soft clauses encodes the reward function R.

$$\bigwedge_{t=1}^{H}\Big(\bigwedge_{i=1}^{n}(r_i^s, Y_{i,t+1}) \wedge \bigwedge_{i=1}^{m}(r_i^a, X_{i,t})\Big) \tag{8}$$

IV. BENCHMARK DOMAIN DESCRIPTIONS

In this section, we provide detailed description of four learned automated planning problems, namely: Navigation [23], Inventory Control [24], System Administrator [25] and Cellda [17].[2]

[2]The repository: https://github.com/saybuser/FD-SAT-Plan

a) Navigation: Navigation [23] task for an agent in a two-dimensional (N-by-N where $N \in \mathbb{Z}^+$) maze is cast as an automated planning problem as follows.

- The location of the agent is represented by N^2 state variables $S = \{s_1, \ldots, s_{N^2}\}$ where state variable s_i represents whether the agent is located at position $i \in \{1, \ldots, N^2\}$ or not.
- The intended movement of the agent is represented by four action variables $A = \{a_1, a_2, a_3, a_4\}$ where action variables a_1, a_2, a_3 and a_4 represent whether the agent attempts to move up, down, right or left, respectively.
- Mutual exclusion on the intended movement of the agent is represented by the global function as follows.
$$C(\langle s_1, \ldots, a_4\rangle) = \begin{cases} true, & \text{if } a_1 + a_2 + a_3 + a_4 \leqslant 1 \\ false, & \text{otherwise} \end{cases}$$
- The initial location of the agent is $s_i = V_i$ for all positions $i \in \{1, \ldots, N^2\}$.
- The final location of the agent is represented by the goal state function as follows.
$$G(\langle s_1, \ldots, s_{N^2}\rangle) = \begin{cases} true, & \text{if } s_i = V_i' \\ & \quad \forall i \in \{1, \ldots, N^2\} \\ false, & \text{otherwise} \end{cases}$$
where V_i' denotes the goal location of the agent (i.e., $V_i' = true$ if and only if position $i \in \{1, \ldots, N^2\}$ is the final location, $V_i' = false$ otherwise).
- The objective is to minimize total number of intended movements by the agent and is represented by the reward function as follows.
$$R(\langle s_1, \ldots, a_4\rangle) = a_1 + a_2 + a_3 + a_4$$
- The next location of the agent is represented by the state transition function T that is a complex function of state and action variables $s_1, \ldots, s_{N^2}, a_1, \ldots, a_4$. The unknown function T is approximated by a BNN $\tilde{T}$, and the details of $\tilde{T}$ are provided in Table I.

We submitted problems with $N = 3, 4, 5$ over planning horizons $H = 4, \ldots, 10$. Note that this automated planning problem is a deterministic version of its original from IPPC2011 [23].

b) Inventory Control: Inventory Control [24] is the problem of managing inventory of a product with demand cycle length $N \in \mathbb{Z}^+$, and is cast as an automated planning problem as follows.

- The inventory level of the product, phase of the demand cycle and whether demand is met or not are represented by three state variables $S = \{s_1, s_2, s_3\}$ where state variables s_1 and s_2 have non-negative integer domains.
- Ordering some fixed amount of inventory is represented by an action variable $A = \{a_1\}$.
- Meeting the demand is represented by the global function as follows.
$$C(\langle s_1, s_2, s_3, a_1\rangle) = \begin{cases} true, & \text{if } s_3 = true \\ false, & \text{otherwise} \end{cases}$$

- The inventory, the phase of the demand cycle and meeting the demand are set to their initial values $s_i = V_i$ for all $i \in \{1, 2, 3\}$.
- Meeting the final demand is represented by the goal state function as follows.

$$G(\langle s_1, s_2, s_3 \rangle) = \begin{cases} true, & \text{if } s_3 = true \\ false, & \text{otherwise} \end{cases}$$

- The objective is to minimize total storage cost and is represented by the reward function as follows.

$$R(\langle s_1, s_2, s_3, a_1 \rangle) = cs_1$$

where c denotes the unit storage cost.
- The next inventory level, the next phase of the demand cycle and whether the next demand is met or not are represented by the state transition function T that is a complex function of state and action variables s_1, s_2, s_3, a_1. The unknown function T is approximated by a BNN $\tilde{T}$, and the details of $\tilde{T}$ are provided in Table I.

We submitted problems with two demand cycle lengths $N \in \{2, 4\}$ over planning horizons $H = 5, \ldots, 8$. The values of parameters are chosen as $m_1 = 4$ and $m_2 = 0$.

c) System Administrator: System Administrator [23], [25] is the problem of maintaining a computer network of size N and is cast as an automated planning problem as follows.

- The age of computer $i \in \{1, \ldots, N\}$, and whether computer $i \in \{1, \ldots, N\}$ is running or not, are represented by $2N$ state variables $S = \{s_1, \ldots, s_{2N}\}$ where state variables $s_1, \ldots, s_N$ have non-negative integer domains.
- Rebooting computers $i \in \{1, \ldots, N\}$ are represented by N action variables $A = \{a_1, \ldots, a_N\}$.
- The bounds on the number of computers that can be rebooted and the requirement that all computers must be running are represented by global function as follows.

$$C(\langle s_1, \ldots, a_N \rangle) = \begin{cases} true, & \text{if } \sum_{i=1}^{N} a_i \leqslant a^{max} \\ & \text{and } s_i = true \\ & \quad \forall i \in \{N+1, \ldots, 2N\} \\ false, & \text{otherwise} \end{cases}$$

where a^{max} is the maximum on the number of computers that can be rebooted at a given time.
- The age of computer $i \in \{1, \ldots, N\}$, and whether computer $i \in \{1, \ldots, N\}$ is running or not are set to their initial values $s_i = V_i$ for all $i \in \{1, \ldots, 2N\}$.
- The requirement that all computers must be running in the end is represented by the goal state function as follows.

$$G(\langle s_1, \ldots, s_{2N} \rangle) = \begin{cases} true, & \text{if } s_i = true \\ & \quad \forall i \in \{N+1, \ldots, 2N\} \\ false, & \text{otherwise} \end{cases}$$

- The objective is to minimize total number of reboots and is represented by the reward function as follows.

$$R(\langle s_1, \ldots, s_{2N}, a_1, \ldots, a_N \rangle) = \sum_{i=1}^{N} a_i$$

- The next age of computer $i \in \{1, \ldots, N\}$ and whether computer $i \in \{1, \ldots, N\}$ will be running or not, are represented by the state transition function T that is a complex function of state and action variables $s_1, \ldots, s_{2N}, a_1, \ldots, a_N$. The unknown function T is approximated by a BNN $\tilde{T}$, and the details of $\tilde{T}$ are provided in Table I.

We submitted problems with $N \in \{4, 5\}$ computers over planning horizons $H = 2, 3, 4$. The values of parameters are chosen as $m_1 = 3$ and $m_2 = 0$.

d) Cellda: Influenced by the famous video game [26], Cellda [17] is the task of an agent who must escape from a two dimensional (N-by-N where $N \in \mathbb{Z}^+$) cell through a locked door by obtaining the key without getting hit by the enemy, and is cast as an automated planning problem as follows.

- The location of the agent, the location of the enemy, whether the key is obtained or not and whether the agent is alive or not are represented by six state variables $S = \{s_1, \ldots, s_6\}$ where state variables s_1 and s_2 represent the horizontal and vertical locations of the agent, state variables s_3 and s_4 represent the horizontal and vertical locations of the enemy, state variable s_5 represents whether the key is obtained or not, and state variable s_6 represents whether the agent is alive or not. State variables s_1, s_2, s_3 and s_4 have positive integer domains.
- The intended movement of the agent is represented by four action variables $A = \{a_1, a_2, a_3, a_4\}$ where action variables a_1, a_2, a_3 and a_4 represent whether the agent intends to move up, down, right or left, respectively.
- Mutual exclusion on the intended movement of the agent, the boundaries of the maze and requirement that the agent must be alive are represented by global function as follows.

$$C(\langle s_1, \ldots, a_4 \rangle) = \begin{cases} true, & \text{if } a_1 + a_2 + a_3 + a_4 \leqslant 1 \\ & \text{and } 0 \leqslant s_i < N \quad \forall i \in \{1, 2\} \\ & \text{and } s_6 = true \\ false, & \text{otherwise} \end{cases}$$

- The location of the agent, the location of the enemy, whether the key is obtained or not, and whether the agent is alive or not are set to their initial values $s_i = V_i$ for all $i \in \{1, \ldots, 6\}$.
- The goal location of the agent (i.e., the location of the door), the requirement that the agent must be alive in the end and the requirement that the key must be obtained are represented by the goal state function as follows.

$$G(\langle s_1, \ldots, s_6 \rangle) = \begin{cases} true, & \text{if } s_1 = V_1' \text{ and } s_2 = V_2' \\ & \text{and } s_5 = true \text{ and } s_6 = true \\ false, & \text{otherwise} \end{cases}$$

where V_1' and V_2' denote the goal location of the agent (i.e., the location of the door).

- The objective is to minimize total number of intended movements by the agent and is represented by the reward function as follows.

$$R(\langle s_1, \ldots, a_4 \rangle) = a_1 + a_2 + a_3 + a_4$$

- The next location of the agent, the next location of the enemy, whether the key will be obtained or not, and whether the agent will be alive or not, are represented by the state transition function T that is a complex function of state and action variables $s_1, \ldots, s_6, a_1, \ldots, a_4$. The unknown function T is approximated by a BNN $\tilde{T}$, and the details of $\tilde{T}$ are provided in Table I.

We submitted problems with maze size $N = 4$ over planning horizons $H = 8, \ldots, 12$ with two different enemy policies. The values of parameters are chosen as $m_1 = 2$ and $m_2 = 0$.

REFERENCES

[1] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[2] H. Kautz and B. Selman, "Planning as satisfiability," in *Proceedings of the Tenth European Conference on Artificial Intelligence*, ser. ECAI'92, 1992, pp. 359–363.

[3] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," in *Journal of Artificial Intelligence Research*, vol. 14. USA: AI Access Foundation, 2001, pp. 253–302.

[4] M. Helmert, "The fast downward planning system," in *Journal Artificial Intelligence Research*, vol. 26. USA: AI Access Foundation, 2006, pp. 191–246.

[5] F. Pommerening, G. Röger, M. Helmert, and B. Bonet, "LP-based heuristics for cost-optimal planning," in *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, ser. ICAPS'14. AAAI Press, 2014, pp. 226–234.

[6] T. O. Davies, A. R. Pearce, P. J. Stuckey, and N. Lipovetzky, "Sequencing operator counts," in *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*. AAAI Press, 2015, pp. 61–69.

[7] W.-M. Shen and H. A. Simon, "Rule creation and rule learning through environmental exploration," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, ser. IJCAI'89. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 675—-680.

[8] Y. Gil, "Acquiring domain knowledge for planning by experimentation," Ph.D. dissertation, Carnegie Mellon University, USA, 1992.

[9] S. W. Bennett and G. F. DeJong, "Real-world robotics: Learning to plan for robust execution," in *Machine Learning*, vol. 23, 1996, pp. 121–161.

[10] S. S. Benson, "Learning action models for reactive autonomous agents," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 1997.

[11] B. Say, G. Wu, Y. Q. Zhou, and S. Sanner, "Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, ser. IJCAI'17, 2017, pp. 750–756.

[12] B. Say and S. Sanner, "Planning in factored state and action spaces with learned binarized neural network transition models," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, ser. IJCAI'18, 2018, pp. 4815–4821.

[13] B. Say, "Optimal planning with learned neural network transition models," Ph.D. dissertation, University of Toronto, Toronto, ON, Canada, 2020.

[14] B. Say, S. Sanner, and S. Thiébaux, "Reward potentials for planning with learned neural network transition models," in *Proceedings of the Twenty-Fifth International Conference on Principles and Practice of Constraint Programming*, T. Schiex and S. de Givry, Eds. Cham: Springer International Publishing, 2019, pp. 674–689.

[15] G. Wu, B. Say, and S. Sanner, "Scalable planning with deep neural network learned transition models," *Journal of Artificial Intelligence Research*, vol. 68, pp. 571–606, 2020.

[16] B. Say, "A unified framework for planning with learned neural network transition models," in *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021, pp. 5016–5024.

[17] B. Say and S. Sanner, "Compact and efficient encodings for planning in factored state and action spaces with learned binarized neural network transition models," *Artificial Intelligence*, vol. 285, p. 103291, 2020.

[18] B. Say, J. Devriendt, J. Nordström, and P. Stuckey, "Theoretical and experimental results for planning with learned binarized neural network transition models," in *Proceedings of the Twenty-Sixth International Conference on Principles and Practice of Constraint Programming*, H. Simonis, Ed. Cham: Springer International Publishing, 2020, pp. 917–934.

[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proceedings of the Thirtieth International Conference on Neural Information Processing Systems*. USA: Curran Associates Inc., 2016, pp. 4114–4122.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the Thirty-Second International Conference on International Conference on Machine Learning*, ser. ICML. JMLR.org, 2015, pp. 448–456.

[21] I. Abío and P. Stuckey, "Encoding linear constraints into SAT," in *Principles and Practice of Constraint Programming*. Springer Int Publishing, 2014, pp. 75–91.

[22] R. Asin and R. Nieuwenhuis, "Cardinality networks and their applications and oliveras, albert and rodriguez-carbonell, enric," in *International Conference on Theory and Applications of Satisfiability Testing*, 2009, pp. 167–180.

[23] S. Sanner and S. Yoon, "International probabilistic planning competition," 2011.

[24] T. Mann and S. Mannor, "Scaling up approximate value iteration with options: Better policies with fewer iterations," in *Proceedings of the Thirty-First International Conference on Machine Learning*, ser. Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32. Bejing, China: PMLR, 2014, pp. 127–135.

[25] C. Guestrin, D. Koller, and R. Parr, "Max-norm projections for factored MDPs," in *Seventeenth International Joint Conferences on Artificial Intelligence*, 2001, pp. 673–680.

[26] Nintendo, "The legend of zelda," 1986.

Benchmark: University Course Timetabling from the International Timetabling Competition 2019

Alexandre Lemos
Instituto Superior Técnico
Universidade de Lisboa, INESC-ID
Lisboa, Portugal
alexandre.lemos@tecnico.ulisboa.pt

Pedro T. Monteiro
Instituto Superior Técnico
Universidade de Lisboa, INESC-ID
Lisboa, Portugal
pedro.tiago.monteiro@tecnico.ulisboa.pt

Inês Lynce
Instituto Superior Técnico
Universidade de Lisboa, INESC-ID
Lisboa, Portugal
ines.lynce@tecnico.ulisboa.pt

***Abstract*—This paper presents a MaxSAT benchmark resulting from a university course timetabling problem. The benchmark was encoded into MaxSAT from the instances of the International Timetabling Competition 2019 [1]. The background on university course timetabling, the formal description of the problem and the full encoding is detailed in the literature [2]–[4].**

The benchmark consists of 30 instances of course timetabling problems from universities all around the world. The benchmark instances were generated without using any of the pre-processing techniques described in the papers. The timetabling problem was split into two sub-problems: course timetabling and student sectioning. The submitted benchmark only has the encoding for the course timetabling sub-problem. There are only 5 instances for which the optimal solution is known.

***Index Terms*—MaxSAT, Benchmark, University Course Timetabling**

I. Problem Overview

A new version of the University Course Timetabling Problem was introduced in the context of the fourth International Timetabling Competition (ITC) 2019 [1]. This problem (hereafter named ITC-2019) can be defined as two complementary sub-problems: (i) course timetabling; and (ii) student sectioning. In this benchmark, we only consider the *course timetabling* sub-problem. The goal is to find a feasible assignment for all the classes of all courses to a time slot and a room, subject to a set of constraints.

II. Maximum Satisfiability

Our work focused on developing a novel MaxSAT based tool *UniCorT*[1] for ITC-2019 that includes the usage of pre-processing techniques to reduce the search space. Our initial solution finished in the TOP 5 of the ITC-2019 competitions.

Our best encoding to solve ITC-2019 utilizes *four* Boolean decision variables: three representing the assignment of a class to a time slot (the week, day, and hour of the class) and one representing the assignment of a class to a room. The ITC-2019 has a rich set of constraints that are mapped to three types of constraints: exactly-one (*e.g.* a class has to be assigned exactly one room), at-least-k (*e.g.* minimum load of classes in a single day) and binary clauses (*e.g.* the classes c_1 and c_2 cannot be taught at the same time).

[1]The tool is available at https://github.com/ADDALemos/MPPTimetables.

The exactly-one and at-least-k constraints are converted into CNF using the totalizer [5] and adder [6] encodings that are available on the solver *TT-Open-WBO-Inc* [7], [8].

III. Benchmark Characteristics

Table I shows the main characteristics of the generated instances. This table summarizes the number of classes and rooms per instance. Furthermore, it shows the number (in thousands) of variables, and hard and soft constraints generated. *UniCorT* is able to find a feasible solution for all instances in this benchmark. However, *UniCorT* is not able to prove optimality in any of these instances. In fact, there are only five instances for which the optimal value is known. These instances are marked in bold in Table I. However, optimality was proven by a mixed-integer programming approach [9], [10][2]. The best-known values can be found at https://www.itc2019.org/home. Note that these values consider the full problem (*i.e.* the student sectioning is also considered).

All the instances in this benchmark are named following the convention:

$$universityName\_(degreeorFaculty)^*\_semesterYear$$

The instances start by the abbreviation of university name and end with the year and semester for which the timetables were extracted. In addition, the instances may also have some information describing a specific degree or faculty. For example, *agh-fis-spr17* is an instance from the spring semester of 2017 for the Faculty of Physics and Applied Computer Science from the university of Akademia Górniczo-Hutnicza.

IV. Parameters Used To Generate Instances

The script used to generate the instances is available at github.com/ADDALemos/MPPTimetables/tree/maxSAT_competition.

For this benchmark, we generated the instances without the sub-problem of student sectioning. Therefore, we do not consider students' conflicts.

Furthermore, the instances are generated before applying the different pre-processing techniques (*e.g.* removal of redundant constraints [3]).

[2]Results of this approach are available at https://dsumsoftware.com/itc2019/.

TABLE I
CHARACTERISTICS OF THE GENERATED INSTANCES. THE INSTANCES FOR WHICH WE KNOW THE OPTIMAL VALUE ARE MARKED IN BOLD.

Instance	# Classes	# Rooms	# Vars (k)	# Hard (k)	# Soft (k)
agh-fal17	5081	327	3,481	55,738	838
agh-fis-spr17	1239	80	230	7,418	10
agh-ggis-spr17	1852	44	1,377	16,997	9
agh-ggos-spr17	1144	84	437	10,342	8
agh-h-spr17	460	39	197	10,424	11
bet-fal17	983	62	561	40,202	40
bet-spr18	1083	63	655	47,975	72
iku-fal17	2641	214	1,937	183,651	176
iku-spr18	2782	208	1,962	190,505	151
lums-fal17	502	97	212	13,924	11
lums-spr18	487	73	203	14,255	10
mary-fal18	951	93	327	15,116	48
mary-spr17	882	90	262	12,075	252
muni-fi-fal17	535	36	111	1,279	2
muni-fi-spr16	575	35	94	1,069	2
muni-fi-spr17	516	35	102	1,237	2
muni-fsps-spr17	561	44	54	480	1
muni-fsps-spr17c	650	29	160	3,427	13
muni-fspsx-fal17	1623	33	580	8,842	23
muni-pdf-spr16	1515	83	1,110	52,849	87
muni-pdf-spr16c	2526	70	2,503	57,036	849
muni-pdfx-fal17	3717	86	7,895	300,398	1,338
nbi-spr18	782	67	175	1,486	3
pu-d5-spr17	1061	84	204	5,125	42
pu-d9-fal19	2798	224	1,057	39,494	219
pu-llr-spr17	1001	75	452	13,400	24
pu-proj-fal19	8813	770	2,704	88,228	159
tg-fal17	711	23	317	2,423	356
tg-spr18	676	24	314	3,460	47
yach-fal17	417	33	76	835	5

All instances were generated with two types of soft constraints: allocation penalty (a cost associated with the assignment of a class to a room or time slot) and soft distribution constraints (*e.g.* a time slot preference).

There is an incremental version of *UniCorT* [4] that at each iteration increases the number of domain options (*i.e.* the number of time slots available to host a class). This version had the best results in terms of performance. Nevertheless, for this benchmark, we generated the instances with all the possible time slots for each class.

V. ACKNOWLEDGMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references SFRH/BD/143212/2019 (PhD grant), DSAIPA/AI/0033/2019 (project LAIfeBlood), DSAIPA/AI/0044/2018 (project Data2help) and UIDB/50021/2020 (INESC-ID multi-annual funding).

REFERENCES

[1] T. Müller, H. Rudová, and Z. Müllerová, "University course timetabling and International Timetabling Competition 2019," in *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan, Eds., 2018, pp. 5–31.

[2] A. Lemos, P. T. Monteiro, and I. Lynce, "Minimal perturbation in university timetabling with maximum satisfiability," in *Proceedings of 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. Springer, 2020, pp. 317–333.

[3] A. Lemos, P. T. Monteiro, and I. Lynce, "Itc 2019: University course timetabling with maxsat," in *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT) Volume 1*, 2020, pp. 105 – 128.

[4] A. Lemos, P. T. Monteiro, and I. Lynce, "Introducing UniCorT: A university course timetabling tool (accepted for publication)," *Journal of Scheduling*, 2021.

[5] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming - CP*, ser. Lecture Notes in Computer Science, F. Rossi, Ed., vol. 2833. Springer, 2003, pp. 108–122.

[6] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.

[7] A. Nadel, "Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization," in *Proceedings of the 19th Conference on Formal Methods in Computer Aided Design (FMCAD)*, 2019.

[8] A. Nadel, "TT-Open-WBO-Inc: Tuning polarity and variable selection for anytime SAT-based optimization," in *Proceedings of the MaxSAT Evaluations*, 2019.

[9] D. S. Holm, R. Ø. Mikkelsen, M. Sørensen, and T. R. Stidsen, "A mip based approach for international timetabling competation 2019," in *Proceedings of the International Timetabling Competition 2019*, 2020.

[10] D. Holm, R. Mikkelsen, M. Sørensen, and T. Stidsen, "A mip formulation of the international timetabling competition 2019 problem," Technical University of Denmark, Tech. Rep., 2020.

Description of Benchmarks on Learning Optimal Decision Trees and Boosted Trees

Hao Hu, Emmanuel Hebrard, Marie-José Huguet, Mohamed Siala
LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
{hhu, hebrard, huguet, siala}@laas.fr

I. Introduction

Decision Trees are one of the most essential models in machine learning, as they are both intrepretable and effective to compute. Unlike traditional top-down heuristic induction for computing decision trees, recently, several exact methods have been introduced to find optimal decision trees via different declarative methods, such as Constraint Programming [4], Boolean Satisfiablity(SAT) [3], and MaxSAT [2]. The objective of the MaxSAT approach is to find decision trees with limited depths that maximize the number of examples correctly classified. It performs better in prediction for unseen data than the SAT approach, as the SAT approach requires perfect accuracy leading to overfitting.

As other exact methods of learning optimal decision trees, the MaxSAT approach also has scalability issues. However, incomplete MaxSAT solvers can produce high quality solutions within a limited time. In addition, the MaxSAT approach can be easily adapted to classic Boosting methods such as AdaBoost [1], to improve the prediction performance. The adaptation is realized by updating the weights of soft clauses corresponding to examples to update the data distribution of each iteration in AdaBoost.

II. MaxSAT Approach of Learning Optimal Decision Trees

A. Problem Definition

The problem solved by the MaxSAT approach is the following optimization problem:

$P(\mathcal{E}, N)$: *Given a set of examples $\mathcal{E}$, find a full binary decision tree of size N that maximizes the number of examples in $\mathcal{E}$ that are correctly classified.*

Since non-binary features can always be transformed as binary features, binary decision trees can handle all data sets. Moreover, to limit the tree depth described in the Introduction, constraints for controlling the size and depth of the tree can be posted in the MaxSAT approach.

To solve $P(\mathcal{E}, N)$, for each example $e_q \in \mathcal{E}$, the MaxSAT approach introduces a Boolean variable b_q, where b_q is true if and only if e_q is correctly classified. Then, all b_q are set as soft clauses and other constraints are set as hard clauses. Therefore, assuming the set of examples $\mathcal{E}$ used is consistent, the unweighted MaxSAT formulation is used.

B. MaxSAT Encoding

The MaxSAT encoding in [2] is largely based on the SAT model from [3] that it extends. The SAT encoding consists of three parts:

- **Part 1**: Constraints on the structure of a valid binary tree in fixed size.
- **Part 2**: Constraints for mapping features (respectively, classes) to internal nodes (respectively, leaf nodes).
- **Part 3**: Constraints for correctly classifying all examples in the example set.

To lift the SAT model into a MaxSAT encoding, for each example e_q, every constraint of **Part 3** concerning e_q is linked to a variable b_q acting as the blocking literal. Then, to achieve the limit in maximum(or exact) depth for decision trees found, two more parts of constraints are added:

- **Part 4**: Constraints for controlling trees in fixed depth.
- **Part 5**: Constraints for controlling tree size under an upper bound not a fixed sized.

Finally, all constraints mentioned are set as hard clauses, and all blocking literal b_q as soft clauses. There is no weight function on the clauses as in this case we try to learn the tree with optimal accuracy and no example is more important than the others.

III. AdaBoost Adaptation

A. AdaBoost Algorithm

Boosting methods are a family of ensemble methods, which train multiple dependent classifiers with the same data set and then combine them to get better predictions than a single classifier. As a typical Boosting method, AdaBoost [1] builds T classifiers in a sequence of T iterations. At each iteration t, AdaBoost learns a classifier h_t and updates the data distribution of the $(t+1)$-th iteration $\mathcal{D}_{t+1}$ based on the t-th data distribution $\mathcal{D}_t$ using the equation 1:

$$\mathcal{D}_{t+1}(x_q) = \frac{\mathcal{D}_t(x_q)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_q) = c_q \\ \exp(\alpha_t) & \text{if } h_t(x_q) \neq c_q \end{cases} \quad (1)$$

In equation 1, each example $e_q = (x_q, c_q)$ is a 2-tuple, where x_q denotes the value vector for all features of this example, and $c_q \in \{0, 1\}$ denotes its class. The coefficient $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$ helps the previously misclassified examples gain more importance in the next iteration, where ϵ_t is the error rate of t-th iteration. Z_t is a normalization factor.

The final predictor is a weighted vote where every classifier h_t is associated with a weight α_t, which is calculated as follows:

$$H(x_q) = 1 \text{ if } \sum_{t=1}^{T} \alpha_t g(h_t(x_q)) > 0 \text{ and } 0 \text{ otherwise} \quad (2)$$

where $g(0) = -1$, $g(1) = 1$. The function H denotes the aggregated predictor.

B. Integration in the MaxSAT Approach

To integrate AdaBoost in the MaxSAT approach, the key idea is to update the data distribution by updating the weights of soft clauses corresponding to examples. The final weighted voting follows the original AdaBoost algorithm in Equation 2.

As the weighted MaxSAT formulation allows only positive integer weights, weights updated from Equation 1 are approximated. We set all weights at the first iteration with the value 1 as initial distribution. Then, two steps of approximation are made to calculate the positive integer weight w_q^{t+1} of soft clause b_q in $(t+1)$-th iteration based on w_q^t, the corresponding weight in previous iteration. Firstly, we update and normalize the weights:

$$\hat{w}_q^{t+1} = \frac{w_q^t * factor_q^t}{\sum_{q=1}^{M}(w_q^t * factor_q^t)} \quad (3)$$

where $factor_q^t$ is the factor based on the prediction:

$$factor_q^t = \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_q) = c_q \\ \exp(\alpha_t) & \text{if } h_t(x_q) \neq c_q \end{cases} \quad (4)$$

Secondly, we discretize the weight $\hat{w}_q^{t+1}$ as follows:

$$w_q^{t+1} = round(\frac{\hat{w}_q^{t+1}}{\min_{i\in\{1,\ldots,M\}}(\hat{w}_i^{t+1})}) \quad (5)$$

IV. Benchmark Instances

There are two first-level folders in the zip archive. The first one is named ***"decision-tree"*** and contains 60 WCNF files that correspond to learning optimal decision trees with 4 different maximum depths for 15 datasets. These benchmarks are suited to the ***unweighted incomplete*** track. The other is named ***"adaboost"*** and contains 120 WCNF files that correspond to learning boosted trees with 4 different maximum depths for 6 datasets in 5 different iterations. These benchmarks are suited to the ***weighted incomplete*** track.

Both first-level folders contains several second-level folders, which correspond to the names of the encoded ML datasets. Each second-level folder contains all WCNF files corresponding to this dataset. The datasets we used to generate WCNF are from CP4IM[1]. More precisely, they are binarized with the *one-hot-encoding*. Since AdaBoost greatly improves the training and test accuracy in some cases, we selected the datasets in which classic decision trees performed poorly to generate WCNF for AdaBoost. Further information on those datasets is given in Table I, where $\#s$ indicates the number of instances, $\#f_b$ indicates the number of binarized features.

TABLE I
INFORMATION OF DATASETS FOR LEARNING OPTIMAL BOOSTED TREES.

Dataset	anneal	australian	car	heart	tumor	tic-tac-toe
$\#s$ / $\#f_b$	812/89	653/124	1728/21	296/95	336/31	958/27

The name of each WCNF file follows the format: formula_*ratio*_*seed*_atleast_*size*_*maxdepth*_reduced_incomplete_*type*.WCNF.

- *ratio*: The sample ratio used when generating a training set using the hold-out method.
- *seed*: The seed used to make the stratified sampling. By default, we use 2021.
- *size*: The upper bound on the size of the decision tree.
- *maxdepth*: The upper bound on the depth of the decision tree.
- *type*: The application of the decision tree generation problem encoded by this WCNF file. There are two possible types:
  - *tree*: This WCNF is for learning a classic decision tree.
  - *adaboost_iter*: This WCNF is for learning decision tree for the (*iter*)-th iteration of AdaBoost.

[1] https://dtai.cs.kuleuven.be/CP4IM/datasets/

References

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997. [Online]. Available: https://doi.org/10.1006/jcss.1997.1504

[2] H. Hu, M. Siala, E. Hebrard, and M. Huguet, "Learning optimal decision trees with maxsat and its integration in adaboost," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, C. Bessiere, Ed. ijcai.org, 2020, pp. 1170–1176. [Online]. Available: https://doi.org/10.24963/ijcai.2020/163

[3] N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva, "Learning optimal decision trees with SAT," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence - IJCAI*, July 13-19, 2018, pp. 1362–1368.

[4] H. Verhaeghe, S. Nijssen, G. Pesant, C. Quimper, and P. Schaus, "Learning optimal decision trees using constraint programming," *Constraints An Int. J.*, vol. 25, no. 3-4, pp. 226–250, 2020. [Online]. Available: https://doi.org/10.1007/s10601-020-09312-3

Partial Weighted MaxSAT Benchmarks: Functional Sequences Maximizing the Sustained Switching Activity in a Pipelined Processor

Nikolaos I. Deligiannis[†], Riccardo Cantoro[†], Tobias Faller[*], Tobias Paxian[*], Bernd Becker[*], Matteo Sonza Reorda[†]
[†] Politecnico di Torino, DAUIN - Torino, Italy [*] Univ. of Freiburg - Freiburg, Germany
{nikolaos.deligiannis|riccardo.cantoro|matteo.sonzareorda}@polito.it {fallert|paxiant|becker}@informatik.uni-freiburg.de

***Abstract*—The benchmarks at hand describe the maximization of switching activity of a certain module of the circuit under test. Maximizing switching activity is beneficial w.r.t. the overall reliability of the device, since it can be utilized during e.g., Burn-In to further exercise the circuit and screen out early failures. We consider the fully pipelined Open RISC 1200 [1] processor for which we were able to constrain the processor state to execute valid instructions according to the OpenRISC 1000 ISA [2]. Additionally, the processor has been constrained to repeatedly execute two instructions that maximize the switching activity of the multiplier contained in the arithmetic and logic unit (ALU).**

***Index Terms*—Switching Activity Maximization, Processor, Formal Techniques, MaxSAT Benchmarks**

I. Introduction

While in most scenarios the minimization of a circuit's switching activity (SWA) is crucial during testing (e.g., to avoid overheating) there are cases (e.g., during Burn-In) where the maximization of the circuit's SWA (or of certain parts of the circuit) can be proven beneficial. Such excitation, which takes place while the circuit is exercised in elevated temperature and power conditions, could further assist to screen out early failures (Infant Mortality) [3].

In this paper we focus on the case where the circuit under test is a fully pipelined processor, namely the Open RISC 1200 (OR1200), and the SWA maximization concerns certain sub-modules of the core. We have devised an algorithm, based on formal techniques, that takes as input the gate level description of the core along with the name of the sub-module we intend to stress and generates sequences of instructions that produce high gate activity within the module. We propose a benchmark suite that is composed of CNFs that correspond to the maximization of the SWA of the multiplier unit of the core. Each benchmark represents a weighted sampling that has been performed on the nets of the targeted module.

II. Benchmark Description

A. Building the CNF(s)

The whole OR1200 processor is translated into a CNF using the Tseitin [4] transformation and is represented by the hard clauses of the problem. Additional hard clauses constrain the processor to be in a running state and the memory interface to only issue two valid and indefinitely repeating instructions on the instruction bus.

The switching of the processor's multiplier gates is encoded as additional variables that are set to true iff the gate's output value changes between the two instructions. Out of all available switching variables only a specified number of variables are randomly chosen and added as soft clauses. The variables are weighted according to the fan-out of the gates and variables of gates with a fan-out of one, e.g. only one following gate, are excluded from the sampling process and are not encoded as soft clauses.

The provided benchmarks comprise of variants with a sample count of 10 to 200 soft clauses. We are mostly interested in solutions to the benchmarks with the highest number of soft clauses, but with the complete MaxSAT Evaluation solver of 2020 we were only able to compute solutions to the small instances (up to 70 soft clauses) in a computation time of one hour.

B. File Name Convention

The file name starts with the problem description followed by the processor name and the number of soft clauses. For example, the proposed MaxSAT benchmark with 50 soft clauses is named:

```
SwitchingActivityMaximization_OpenRISC1200_50.wcnf
```

References

[1] D. Lampret, "Openrisc 1200 ip core specification rev. 0.7," 9 2001. [Online]. Available: https://opencores.org/websvn/filedetails?repname=or1k&path=%2For1k%2Ftrunk%2Fdocs%2Fopenrisc1200_spec.pdf

[2] D. Lampret, C.-M. Chen, M. Mlinar, J. Rydberg, M. Ziv-Av, C. Ziomkowski, G. McGary, B. Gardner, R. Mathur, M. Bolado, Y. Vernier, J. Baxter, and S. Kristiansson, "Openrisc 1000 architecture manual, architecture version 1.1, document revision 0," 4 2014. [Online]. Available: https://openrisc.io/or1k.html

[3] T. Mak, "Infant Mortality–The Lesser Known Reliability Issue," in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, 2007, pp. 122–122.

[4] G. Tseitin, "On the Complexity of Derivation in Propositional Calculus," *Studies in Constructive Mathematics and Mathematical Logic*, 1968.

Solver Index

Benchmark Index

Author Index